Algorithmes d'analyse syntaxique par grammaires lexicalisées : optimisation et traitement de l'ambiguïté

Olivier Blanc

THÈSE DE DOCTORAT

en Linguistique-informatique

Algorithmes d'analyse syntaxique par grammaires lexicalisées : optimisation et traitement de l'ambiguïté

Olivier Blanc

Université de Marne-la-Vallée

Université de Marne-la-Vallée

THÈSE DE DOCTORAT

en Linguistique-informatique

Algorithmes d'analyse syntaxique par grammaires lexicalisées : optimisation et traitement de l'ambiguïté

Olivier Blanc

Directeur de thèse
Eric Laporte

Présentée et soutenue publiquement le
8 Décembre 2006

Devant le jury composé de :
Laurence Danlos (Présidente de séance)
Cédrick Fairon (Examinateur)
Franz Guenthner (Rapporteur)
Eric Laporte (Directeur de thèse)
Denis Maurel (Rapporteur)

à Pierre et Pierrette

Remerciements

Je voudrais d'abord remercier Eric Laporte pour m'avoir donner l'oportunité de faire cette thèse, pour la confiance qu'il a porté à mon travail ainsi que pour la grande liberté qu'il m'a donné dans l'orientation de mes recherches.

Je remercie très fort les professeurs Franz Guenthner et Denis Maurel pour avoir accepté d'être rapporteurs de cette thèse, pour leur enthousiasme et leurs commentaires pertinents sur mon travail. Un très grand merci aussi à Laurence Danlos et à Cédrick Fairon de m'avoir fait l'honneur et le bonheur d'être membres de mon jury de thèse.

Je remercie l'Institut Gaspard-Monge de m'avoir mis à disposition tous les moyens nécessaires à l'accomplissement de ce travail. Je remercie l'ensemble de l'équipe Informatique-linguistique et ses nombreux thèsards qui m'ont accompagné durant ces quatres années.

Je tiens également à remercier Tita Kyriacopoulou et toute son équipe pour leur accueil toujours chaleureux lors de mes séjours dans les sous-sols de Thessaloniki. Je remercie aussi l'équipe du CENTAL de Louvain-la-Neuve pour leur accueil tout aussi exceptionnel. Un merci en particuler à Anne Dister pour toutes ses petites attentions et à Patrick Watrin pour ses encouragements et son enthousiasme envers mon travail. J'espère avoir la chance de collaborer à nouveau avec toutes ces personnes dans l'avenir.

Je remercie toute ma famille pour son soutien, et ma mère qui a eu le courage de relire l'intégralité de ce mémoire à la recherche des nombreuses coquilles.

Je remercie Matthieu Constant pour nos fructueuses collaborations internationales et scientifiques et pour avoir fait la première relecture "au compte goutte" de ce manuscript. Je remercie Takuya Nakamura pour sa présence et son soutien au quotidien tant au niveau professionnel que personnel.

Résumé

Nos recherches portent sur l'analyse automatique de textes par application de grammaires lexicalisées en utilisant des ressources linguistiques à large couverture. Dans ce contexte, nous avons appronfondi nos travaux dans trois domaines : l'algorithmique, la réalisation d'applications utilisables dans un contexte industriel et l'analyse syntaxique profonde. En ce qui concerne le premier point, nous avons implémenté des algorithmes originaux pour l'optimisation des grammaires locales en préalable à leur utilisation pour l'analyse et nous proposons un algorithme efficace pour l'application de ce type de grammaire sur un texte. Notre algorithme améliore le traitement des ambiguïtés lexicales et syntaxiques. Nous montrons par des évaluations chiffrées que nos algorithmes permettent de traiter de gros volumes de données textuelles en combinaison avec des ressources linguistiques fines et à large couverture. Au niveau applicatif, nous avons participé au développement de la plate-forme RNTL Outilex dédiée aux traitements automatiques de textes écrits. L'architecture modulaire de la plate-forme et sa licence peu restrictive (LGPL) permet la réalisation, à faible coût, d'applications hybrides mélangeant les méthodes à base de ressources linguistiques avec les méthodes statistiques. Enfin, le troisième axe de nos recherches porte sur l'exploitation des tables du lexique-grammaire, pour l'analyse syntaxique profonde et l'extraction des prédicats et de leurs arguments dans les textes français. A cet effet, nous avons fait évoluer le formalisme de nos grammaires vers un formalisme à structure de traits. Les équations sur les traits qui décorent notre grammaire nous permettent de résoudre de manière déclarative différents phénomènes syntaxiques et de représenter de manière formelle les résultats d'analyse. Nous présentons notre grammaire du français dans l'état actuel, qui est générée de manière semi-automatique à partir des tables du lexique grammaire, et nous donnons des évaluations de sa couverture lexicale et syntaxique.

Abstract

The present work is about automatic parsing of written texts using lexicalized grammars and large coverage language resources. More specifically, we concentrated our work on three domains : algorithmic, easy development of NLP applications useful in an industrial context, and deep syntactic parsing. Concerning the first point, we implemented new algorithms for the optimisation of local grammars before their use for parsing and we propose an efficient algorithm for the application of this kind of grammar on text. Our algorithm enhance the processing of lexical and syntactic ambiguities and we show that it scales well when processing big text corpora in combination with fine grained and large coverage language resources.

Concerning the second point, we were actively commited to the development of the Outilex project, a generalist linguistic platform dedicated to text processing. By its modular architecture, the platform aims to provide easy development of high level hybrid NLP applications mixing symbolic and stochastic approachs.

Finally, the third part of our researchs involves the exploitation of the lexicon-grammar tables for deep syntactic parsing and the identification of predicate-arguments structures in French texts. For this purpose, we enhanced the formalism of local grammars with the addition of features structure constraints. Those constraints make possible to declaratively solve in our grammar many syntactic phenomena and to formalize the result of syntactic parsing. We present our grammar for French in its current state, which is semi-automatically generated from the lexicon-grammar tables, and we show some evaluation of its lexical and syntactic coverage.

Table des matières

Table des figures

Chapitre 1

Introduction

Les travaux décrits dans ce mémoire traitent de l'analyse automatique de textes écrits. Cette problématique est devenue aujourd'hui incoutournable, du fait notamment de l'explosion du nombre de documents nouveaux produits et disponibles chaque jour, grâce aux développements de l'informatique et d'internet. Il est ainsi devenu nécessaire de développer des applications nouvelles permettant de traiter automatiquement et efficacement ces quantités importantes de données textuelles. Les applications sont multiples [Pierrel, 2000] : classification automatique de documents, interrogation de bases de données textuelles, génération de résumés, extraction d'informations ciblées, traduction assistée et automatique, etc.

La grande majorité des applications courantes du domaine fonctionnent aujourd'hui à partir de méthodes stochastiques avec entraînement sur des grands corpus, et utilisent peu ou pas de ressources linguistiques [Schmid, 1994] [Brill, 1995] [Church, 1988] [Rajman, 1995]. Ces approches statistiques ont l'avantage de donner des résultats très vite et à faible coût. Cependant les modèles sous-jacents sont souvent très simples et échouent à représenter la complexité des langues naturelles, ce qui cause des taux d'erreur non négligeables dans les résultats d'analyse. Ainsi de nombreux chercheurs s'accordent à dire que de tels systèmes basés uniquement sur des méthodes statistiques devraient vite atteindre leurs limites et qu'il est nécessaire, pour les dépasser, de prendre en compte le contenu linguistique des données traitées

[Gross et Senellart, 1998] [Abeillé et Blache, 2000]. Ainsi, toute une partie de la communauté s'intéresse à l'utilisation d'approches symboliques en exploitant les résultats de la linguistique formelle. De tels systèmes, pour être réellement utilisables en pratique, nécessitent des ressources linguistiques fines et à large couverture (dictionnaires et grammaires) dont la réalisation a un coût non négligeable.

Ces approches utilisant des descriptions linguistiques formalisées ont l'avantage de faire une distinction nette entre les programmes et les ressources linguistiques utilisées. En particulier, elles utilisent des grammaires lisibles, qui peuvent être écrites et modifiées par des linguistes non informaticiens. Les formalismes grammaticaux utilisés en TAL sont nombreux et se distinguent par leur niveau d'expressivité. Les formalismes de plus bas niveau consistent en des outils purement formels et dépourvus de toute notion linguistique : les automates finis, les grammaires hors-contexte et les réseaux de transitions recursifs par exemple. De tels outils ont l'avantage d'avoir de très bonnes performances en termes de temps d'exécution et sont souvent utilisés pour des traitements qui ne nécessitent qu'une analyse partielle des textes en entrée [Roche et Schabes, 1997]. De plus, ils sont indépendants de la théorie linguistique sous-jacente et peuvent donc être utilisés avec des théories linguistiques différentes. Cependant, la simplicité de ces modèles les rend mal adaptés pour l'analyse syntaxique profonde d'énoncés dans leur intégralité. Ainsi, des formalismes grammaticaux spécifiques à l'analyse de textes en langues naturelles ont été inventés comme les grammaires de clauses définies (DCG) [Pereira et Warren, 1980] ou PATR-II [Shieber, 1986]. Dans ces formalismes, les règles de réécriture sont enrichies par des structures de traits qui se combinent par des opérations d'unification et qui spécifient des contraintes additionnelles que les mots doivent vérifier pour que la séquence en entrée soit acceptée par la grammaire. Ces travaux ont donné naissance à des formalismes de plus haut niveau basés sur l'unification et incorporant une véritable théorie linguistique [Abeillé, 1993]. Ils offrent un bon compromis entre la facilité de description linguistique et l'implémentation d'algorithmes efficaces pour l'analyse automatique.

Dans ce mémoire, nous nous intéressons tout d'abord au formalisme des grammaires locales [Gross, 1993]. Ce formalisme a été initialement développé avec le système Intex [Silberztein, 1993] et a été adopté comme outil de représentation grammatical par l'ensemble de la communauté RELEX dont

le laboratoire de l'IGM fait partie. Ces grammaires se présentent sous la forme de graphes lexicalisés et permettent de représenter et localiser de manière très précise dans les textes des constructions locales comme les dates [Maurel, 1990], les déterminants numéraux [Silberztein, 1993][Chrobot, 2000] ou les incises [Fairon, 2000] par exemple. Les grammaires locales sont traditionnellement présentées comme équivalentes à des automates à états finis (ou, si les appels récursifs sont autorisés, à des réseaux de transitions recursifs), mais nous montrons que certaines propriétés sur leurs étiquettes et leur alphabet d'entrée font que les deux modèles ne sont pas strictement équivalents. Nous proposons alors de représenter ce type de grammaire par des automates lexicaux, une variante plus spécialisée du modèle des automates finis dans lequel les transitions sont étiquetées par des symboles ensemblistes. Nous présentons des algorithmes d'optimisation de ce type de grammaires qui nous ont permis d'obtenir des gains sensibles en performance lors de leur application pour l'analyse de textes. Nous nous servons du modèle des automates lexicaux pour représenter nos grammaires mais également pour représenter les textes à l'issue de la phase d'étiquetage morpho-syntaxique, la représentation du texte par un automate acyclique permettant de rendre compte des ambiguïtés d'étiquetage et de découpage. Ainsi, l'application d'une grammaire à un texte se résume à une opération sur deux automates lexicaux. Nous présentons certaines de ces opérations comme la levée d'ambiguïtés lexicales par intersection d'automates, ou l'annotation de textes par l'application d'un transducteur lexical.

La seconde partie, plus applicative, est consacrée à la présentation du projet de plate-forme RNTL Outilex. Ce projet vise à mettre à la disposition de la recherche, du développement et de l'industrie une plate-forme logicielle ouverte dédiée au traitement automatique de textes. Dans ce contexte, nous avons été amené à développer diverses applications implémentant les opérations fondamentales pour le traitement automatique de textes écrits. Nous présentons ainsi les principaux résultats de la plate-forme, et les composants logiciels que nous avons développés dans le cadre de ce projet. Nous présentons plus en détail les modules concernant les traitements par grammaires dans le formalisme des WRTN, qui consiste en une extension du formalisme des grammaires locales dans lequel nous avons intégré une composante de pondération sur les transitions. Ce système de pondération devrait faciliter la réalisation de sytèmes hybrides mélangeant les approches statistiques et celles utilisant des ressources linguistiques.

Dans la troisième partie de ce mémoire, nous présentons nos recherches sur l'exploitation des propriétés syntaxiques et transformationnelles décrites dans les tables du lexique-grammaire [Leclère, 1990] pour l'analyse syntaxique profonde et l'extraction des phrases simples (identification des prédicats et de leurs arguments) dans les textes français. A cet effet, nous avons enrichi le formalisme des WRTN d'une composante d'unification, pour le faire évoluer vers un formalisme original à structure de traits : les RTN décorés (DRTN). Nous conservons la simplicité initiale du système de graphes, tout en augmentant notre grammaire d'équations sur les traits qui nous permettent, d'une part, de résoudre différents phénomènes linguistiques (tels que les contraintes d'accord, les phénomènes d'extraction et les dépendances à distance ou encore la résolution de certaines coréférences) et, d'autre part, de coder les fonctions grammaticales des actants syntaxiques. Nous détaillons d'abord les propriétés du formalisme DRTN et nous présentons les mécanismes qui nous permettent de générer la grammaire lexicalisée du français à partir d'une meta grammaire construite manuellement. Puis, nous présentons la constitution de notre grammaire en l'état actuel et nous donnons des évaluations de sa couverture syntaxique et lexicale.

Nous terminons par une conclusion générale, qui résume les principaux résultats de nos travaux et nous donnons quelques pistes vers lesquelles nous souhaitons poursuivre nos recherches à l'avenir.

Chapitre 2

Automates lexicaux

Les automates et transducteurs finis ont prouvé leur utilité dans une large variété d'applications en informatique linguistique. Ils permettent par exemple une représentation compacte et à accès rapide pour les lexiques à large couverture [Revuz, 1991] et sont à la base d'algorithmes efficaces à de nombreuses étapes du traitement des langues naturelles, de l'analyse phonologique et morphologique [Kaplan et Kay, 1994, Karttunen *et al.*, 1992] et la reconnaissance de la parole [Mohri, 1997] jusqu'à l'analyse syntaxique de texte [Roche et Schabes, 1997].

Dans cette partie, nous nous intéressons plus particulièrement au modèle des grammaires locales tel que présenté dans [Gross, 1997]. Ce formalisme grammatical permet de faire des descriptions sous forme de graphes de formes linguistiques complexes relevant d'un même domaine syntaxique [Fairon, 2000] ou sémantique [Constant, 2000] ; il a été adopté dans de nombreuses recherches en traitement automatique des langues par l'intermédiaire des logiciels INTEX [Silberztein, 1993] et Unitex [Paumier, 2003b]. Une grammaire locale est traditionnellement définie comme un automate à états finis. Cependant, les deux modèles ne sont pas équivalents. En particulier, la définition formelle de certaines notions telle que le déterminisme diffère entre les deux modèles et certains algorithmes de la théorie des automates finis (tels que la déterminisation ou l'intersection de deux automates finis) n'ont pas un comportement correct lorsqu'ils sont appliqués directement sur les grammaires

locales. Nous présentons, dans ce chapitre, le modèle des automates lexicaux, qui consiste en une extension du modèle des automates finis, dans lequel les transitions sont étiquetées par des masques lexicaux. Les masques lexicaux sont des étiquettes ensemblistes permettant de représenter de manière uniforme et structurée les différents symboles qui étiquettent les grammaires locales. Nous montrons comment nous sommes parvenus à implémenter diverses opérations sur les automates lexicaux, telles que la déterminisation, l'intersection ou la complémentation.

Les automates lexicaux permettent de représenter les grammaires locales mais nous nous en servons également pour la représentation des corpus de texte après la phase d'étiquetage morpho-syntaxique, la structure d'automate permettant de représenter les ambiguïtés d'étiquetage et de découpage. Ainsi, l'application d'une grammaire locale à un texte se résume à une opération sur deux automates lexicaux. Nous présentons certaines de ces opérations telles que la levée d'ambiguïtés lexicales par intersection d'automates ou l'annotation de texte par application d'un transducteur lexical.

2.1 Les grammaires locales

Les logiciels INTEX et Unitex intègrent un éditeur de graphes permettant la construction de grammaires locales. Dans ces environnements, les états sont laissés implicites et les symboles de l'alphabet d'entrée sont représentés dans des boites connectées entre elles par des arcs non étiquetés. Ces grammaires sont ensuite traduites sous la forme d'automates pour faciliter les traitements informatiques. La figure 2.1, par exemple, présente un graphe simple qui décrit un verbe conjugué à l'un des 8 temps de l'indicatif. Les 4 temps simples sont indiqués par les codes P (présent), I (imparfait), J (passé simple) et F (futur) ; les 4 temps composés sont décrits dans le chemin du haut passant par l'auxiliaire *être* ou *avoir* conjugué suivi du verbe au participe passé (code K). La figure 2.2 présente la même grammaire dans sa représentation duale sous la forme d'un automate fini.

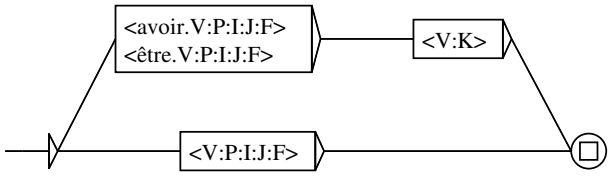

FIG. 2.1 – Grammaire locale sous forme de graphe

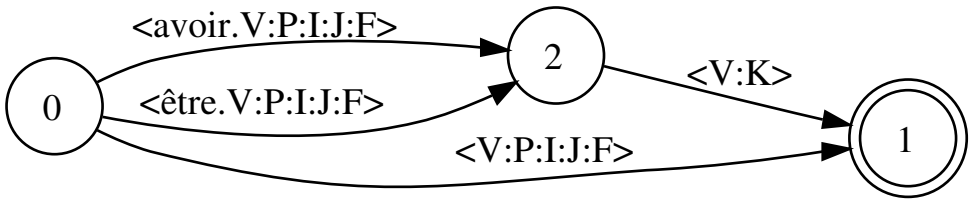

FIG. 2.2 – Représentation duale

Les symboles qui étiquettent les transitions de la grammaire décrivent des unités lexicales (ou des tokens). Ces étiquettes peuvent être de plusieurs formes :

soit une forme fléchie :
Par exemple l'étiquette *avions* reconnaîtra toutes les occurrences de la graphie *avions* dans un texte (qu'il s'agisse d'une forme conjuguée du verbe *avoir* ou du substantif) ;

soit une étiquette lexicale incomplète :
Dans ce type d'étiquette sont spécifiés la catégorie grammaticale ainsi qu'un ensemble de codes morpho-syntaxiques. L'étiquette peut également comporter un lemme. Par exemple :
– l'étiquette <avoir.V :P :I :J :F>, utilisée dans la grammaire (figure 2.1), reconnaît toutes les formes du verbe *avoir* conjugué à l'un des 4 temps simples de l'indicatif ;

– l'étiquette <N+Conc> décrit l'ensemble des *noms concrets* ;

soit un symbole *spécial* :

Les manuels d'Intex et Unitex parlent de symboles META :

– le symbole <E> représente le mot vide ; il reconnaît la séquence vide ;
– <TOKEN> : reconnaît n'importe quelle unité lexicale (mot, nombre ou symbole de ponctuation) ;
– <MOT> : reconnaît n'importe quel mot ;
– <MIN> : reconnaît n'importe quel mot en minuscules ;
– <MAJ> : reconnaît n'importe quel mot en majuscules ;
– <PRE> : reconnaît n'importe quel mot commençant par une majuscule ;
– <PNC> : reconnaît n'importe quel symbole de ponctuation ;
– <NB> : reconnaît n'importe quel nombre (séquence de chiffres) ;
– <DIC> : reconnaît n'importe quel mot ayant reçu une analyse par consultation des dictionnaires ;
– <!DIC> : reconnaît les mots inconnus, c'est-à-dire les formes attestées dans les textes mais qui n'ont pas d'analyse dans les dictionnaires.

Ainsi, les symboles qui étiquettent les transitions d'une grammaire locale sont de types hétérogènes. D'autre part, ce sont des symboles ensemblistes, dans le sens où une étiquette peut reconnaître un ensemble d'unités lexicales. L'ensemble décrit par certains symboles, tels que <NB> ou <!DIC> par exemple, peut même être de cardinal non borné. Du fait de ces caractéristiques, le modèle diffère substantiellement du modèle des automates finis tel qu'il est traditionnellement défini en théorie des langages (dans [Hopcroft et Ullman, 1979] par exemple). En effet, les définitions formelles de certaines notions fondamentales, telles que le *déterminisme*, sont modifiées. Formellement, un automate fini est dit déterministe s'il a au plus un unique état initial et si pour chaque symbole de l'alphabet d'entrée, il a pour tout état au plus une transition sortante étiquetée par ce symbole. L'utilisation d'automates déterministes permet souvent d'obtenir des traitements optimaux puisque leur comportement est déterminé, à chaque étape du calcul, de façon unique pour toute entrée. Or, si nous prenons l'automate de la figure 2.2, même si ces propriétés sont vérifiées, l'automate ne peut pas pour autant être considéré comme déterministe puisque, lorsqu'il est positionné à l'état 0, la lecture du mot étiqueté {ai,avoir.V :P1s} peut positionner l'automate dans l'état 1 ou

l'état 2. Ceci est dû au fait que deux symboles différents peuvent avoir une intersection non vide. Cette propriété a des conséquences sur la déterminisation mais également sur la plupart des algorithmes de manipulation des automates, tels que la complémentation ou le calcul de l'intersection de deux automates ; les algorithmes standards ne peuvent pas être utilisés.

D'autre part, cette hétérogénéité sur les types de symboles qui peuvent étiqueter une grammaire locale implique des traitements spécifiques en fonction du type d'étiquettes lors des différents traitements qui sont opérés sur les grammaires, et lors de leur confrontation avec des textes. Ainsi, dans [Paumier, 2000] puis [Paumier, 2003c], S. Paumier propose des méthodes d'optimisation spécifiques à ce type de grammaires. Ces méthodes mettent en oeuvre des pré-calculs qui diffèrent selon la nature des étiquettes utilisées dans les grammaires et qui dépendent étroitement du vocabulaire du texte analysé, ce qui implique notamment que l'optimisation d'une grammaire doit être à chaque fois recalculée pour l'analyse d'un nouveau texte.

Nous avons opté pour une approche fondamentalement différente pour le traitement des grammaires et leur utilisation pour l'analyse. En effet, plutôt que d'essayer de tirer parti des particularités sur les étiquettes pour accélérer l'analyse, nous avons simplifié le modèle grammatical formel. Pour ce faire, nous avons unifié les différents types que peut avoir une étiquette vers une unique forme de représentation, que nous appelons *masque lexical*.

2.2 Masque lexical

Le masque lexical reprend le concept d'*étiquette incomplète* décrit dans la section précédente : il s'agit d'une étiquette lexicale structurée dans laquelle toutes les informations ne sont pas nécessairement représentées. Ainsi, un masque lexical décrit l'ensemble des mots étiquetés qui vérifient les traits qui lui sont spécifiés. Ces traits peuvent porter sur le lemme, la forme fléchie, la catégorie grammaticale de l'entrée ou sur ses codes morpho-syntaxiques et flexionnels. Contrairement aux étiquettes incomplètes utilisées dans les grammaires au formalisme INTEX/Unitex (dans lesquels les traits consistent en une séquence de codes non typés), dans un masque lexical, avec nos conven-

tions les traits peuvent être représentés sous la forme de couples attribut-valeur. Ainsi par exemple :

– le masque <verb+tense=P+person=3+number=s> reconnaît l'ensemble des verbes conjugués à la troisième personne du singulier du présent de l'indicatif.

Dans un masque lexical, la valeur d'un trait pour un attribut donné n'est pas nécessairement constituée d'un symbole atomique mais peut-être sous la forme d'une disjonction de symboles :

– le masque <noun+subcat=hum|conc> reconnaît l'ensemble des substantifs appartenant à la sous-catégorie des noms humains ou concrets. Cet ensemble ne peut être exprimé dans le formalisme d'INTEX/Unitex que par une disjonction de deux étiquettes incomplètes : <N+Hum>+<N+Abst>.

Il est également possible de nier dans un masque lexical une liste de formes fléchies ou canoniques :

– le masque <femme|homme.noun> reconnaît l'ensemble des formes fléchies des substantifs *femme* et *homme* ;
– le masque <!femme!homme.noun+subcat=hum> par exemple reconnaît l'ensemble des substantifs appartenant à la sous-catégorie des *noms humains*, excepté les formes ayant *homme* ou *femme* pour lemme : sont ainsi reconnus *fille*, *scaphandriers* ou *metteur en scène*, mais ni *femmes* ni *réverbère*.

Comme nous le verrons par la suite, nous avons mis en place des mécanismes de raccourcis permettant au grammairien d'utiliser directement les notations traditionnelles. Par exemple le masque sous la forme canonique : <verb+tense=P+person=3+number=s> peut également s'écrire <V :P3s>.

Nous avons fait en sorte que l'ensemble des symboles utilisés dans les grammaires locales (graphies sans code grammatical, étiquettes incomplètes et symboles META) puissent être représentés de façon uniforme et structurée par un masque lexical. De cette manière, nous obtenons une parfaite interopérabilité entre les différents systèmes, ce qui nous a permis de réutiliser sans modification l'ensemble des grammaires existantes qui ont été écrites

pour les systèmes INTEX et Unitex.

2.3 Description formelle du jeu d'étiquettes

La composition des masques lexicaux est paramétrée par une description formelle du jeu d'étiquettes utilisé. Cette description consiste en une énumération de l'ensemble des catégories grammaticales, avec pour chacune d'entre-elles, la liste des traits flexionnels ou syntaxiques qui peuvent étiqueter une entrée appartenant à cette catégorie.

L'utilisation d'une description extérieure à nos programmes nous a permis d'implémenter nos algorithmes de manipulation des masques lexicaux indépendamment du jeu d'étiquettes utilisé. Ceci nous a permis d'utiliser nos programmes avec des dictionnaires de richesses variées et surtout de traiter des textes dans d'autres langues que le français.

La description du jeu d'étiquettes se fait dans un fichier XML. Nous présentons dans la figure 2.3 un extrait du fichier *lingdef.xml* que nous utilisons pour décrire le jeu d'étiquettes des dictionnaires DELA du français [Courtois, 1990][Courtois, 2004]. Le fichier complet est présenté dans les annexes.

```
<lingdef lang='fr'>
<attrtype name="gender" type='enum'>
  <value name='m' alias='masculine'/>
  <value name='f' alias='feminine'/>
</attrtype>
<attrtype name='nounsubcat' type='enum'>
  <value name='pred' alias='Pred'/>
  <value name='abst' alias='Abst,abstract,abs'/>
  <value name='conc' alias='Conc,concret'/>
  <value name='hum'  alias='Hum,human'/>
  <value name='anl'  alias='Anl,animal'/>
  <value name='tps'  alias='Tps,temporal'/>
  <value name='top'  alias='Top,toponym'/>
  <value name='unit' alias='Unit'/>
  <value name='num'  alias='Nnum,numeral'/>
  <value name='dnom' alias='Dnom,detnom'/>
</attrtype>
<attrtype name='proper' type='bool'>
  <true alias='pr,Pr,Prenom'/>
</attrtype>
[...]
<pos name='det' cutename='DET'>
  <attribute name='subcat'    type='detsubcat' shortcut='yes'/>
  <attribute name='gender'    type='gender'    shortcut='yes'/>
  <attribute name='number'    type='number'    shortcut='yes'/>
  <attribute name='gennumber' type='number'    shortcut='no'/>
  <attribute name='genperson' type='person'    shortcut='no'/>
</pos>
<pos name='noun' cutename='N'>
  <attribute name='subcat' type='nounsubcat' shortcut='yes'/>
  <attribute name='gender' type='gender' shortcut='yes'/>
  <attribute name='number' type='number' shortcut='yes'/>
  <attribute name='proper' type='proper' default='false' shortcut='yes'/>
  <attribute name='compound' type='compound' default='false' shortcut='yes'/>
  <attribute name='coll'   type='collective' shortcut='yes'/>
</pos>
</lingdef>
```

FIG. 2.3 – Extrait de la description du jeu d'étiquettes pour le français

Un document XML de type *lingdef* est constitué d'une séquence d'éléments de type *attrtype* et *pos*. Les éléments *attrtype* servent à la définition de nouveaux types d'attributs ; les éléments *pos* permettent de déclarer les différentes catégories grammaticales.

2.3.1 Définition des types d'attribut

L'élément XML *attrtype* est utilisé pour définir un nouveau type d'attribut. A chaque type d'attribut est associé un identifiant (attribut *name*) ainsi qu'un type (attribut *type*). Ce type peut être soit *bool*, soit *enum*.

- Le type *bool* permet de représenter des attributs à valeur booléenne (par exemple, l'attribut *proper* qui indique pour un nom s'il s'agit d'un nom propre) ;
- le type *enum* est utilisé pour les attributs qui peuvent prendre une valeur parmi une liste bien définie (par exemple l'attribut *gender* peut prendre la valeur *masculine* ou *feminine*) ;

Lorsque l'attribut est de type *enum*, les éléments *value* qui composent l'élément *attrtype* indiquent les différentes valeurs que peut prendre l'attribut : le champ *name* indique un nom canonique pour chacune de ces valeurs et le champ *alias* présente une liste de noms alternatifs. Le nom canonique ou les noms alternatifs peuvent être utilisés et interchangés sans aucune conséquence dans les grammaires. Ainsi, d'après notre extrait, les masques <noun+gender=feminine> et <noun+gender=f> sont strictement équivalents.

De la même manière, il est possible de spécifier des alias pour les valeurs *true* et *false* des attributs de type booléen. Ainsi, toujours d'après notre extrait, le masque <noun+proper=true> peut également être écrit <noun+proper=pr>.

2.3.2 Définition des catégories grammaticales

L'élément *pos* (pour part-of-speech) permet de déclarer une nouvelle catégorie grammaticale. Chaque catégorie a un nom (spécifié dans l'attribut *name*) et éventuellement une liste de noms alternatifs (spécifiés dans l'attribut *cutename*). La description des traits morpho-syntaxiques qui peuvent étiqueter un mot appartenant à la catégorie grammaticale en question se fait à l'aide des éléments XML de type *attribute*. A chacun de ces attributs est associé un nom (attribut *name*) et un type (attribut *type*) qui fait référence à l'identifiant d'un type d'attribut précédemment défini.

Notons que le fait de distinguer le nom d'un attribut de son type nous permet d'utiliser un même type d'attribut pour représenter différents traits au sein d'une même catégorie grammaticale. Ainsi, dans la définition de la catégorie *det*, le type d'attribut *number* est utilisé pour représenter le nombre du déterminant (attribut *number*) ainsi que le nombre du *possesseur* (attribut *gennumber*) pour le cas des déterminants possessifs (*mon*, *ton*, *son*, etc.). Parallèlement, cette distinction nous permet d'utiliser un même nom pour représenter des traits qui ne sont pas du même type dans des catégories grammaticales différentes : par exemple, le nom *subcat* est utilisé dans la description des noms et des déterminants pour définir des traits de types différents (*nounsubcat* et *detsubcat* respectivement).

Lorsque la valeur de l'attribut *shortcut* est positionnée à *yes*, il est possible d'utiliser la notation raccourcie (+valeur) en lieu et place de la notation longue (+attr=valeur) pour l'écriture des masques lexicaux dans les grammaires. Ainsi, d'après notre description de la figure 2.3, il est possible d'écrire dans une grammaire le masque
<noun+subcat=hum+gender=m+number=p>
sous la forme abrégée : <N+hum+m+p>.

De plus, lorsque la valeur d'un trait n'est constituée que d'une seule lettre, il est également possible de l'écrire dans un quatrième champ du masque lexical séparé du champ précédent par le symbole :. Ainsi, le même masque peut s'écrire : <N+Hum :mp>. Le traitement de ce type de notation permet une meilleure interopérabilité avec les grammaires existantes écrites pour les logiciels INTEX et Unitex.

2.4 Catégories grammaticales spéciales

La description du jeu d'étiquettes *lingdef* nous permet d'obtenir une représentation structurée des étiquettes grammaticales associées aux entrées des dictionnaires utilisés lors de la phase d'étiquetage morpho-syntaxique. Dans la pratique, tous les tokens qui sont susceptibles d'apparaître dans un texte ne sont pas nécessairement des entrées d'un dictionnaire. Nous pensons notamment aux nombres en chiffres, aux différents symboles de typographie et ponctuation ainsi qu'aux mots inconnus absents des dictionnaires. Afin de pouvoir traiter de manière uniforme tous les tokens qui apparaissent dans les textes, nous avons établi des *catégories grammaticales spéciales* spécifiques à chacun de ces types de token. De cette manière, nous pouvons représenter par un masque lexical l'ensemble des tokens qui apparaissent dans les textes. La figure 2.4 présente un exemple d'automate de phrase dans lequel sont présentées la plupart de ces catégories grammaticales spéciales.

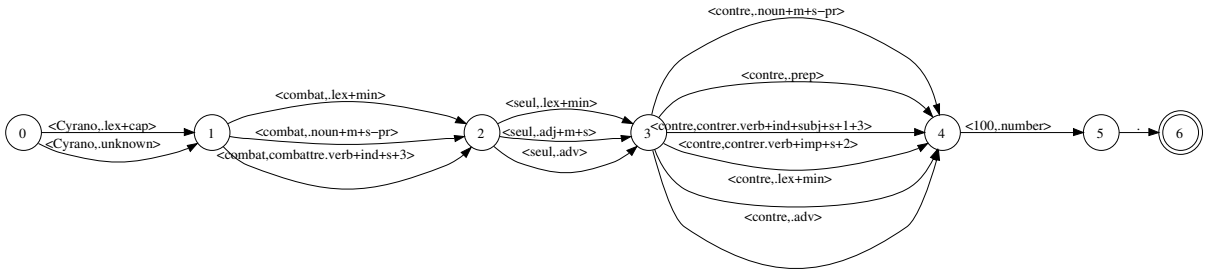

FIG. 2.4 – Catégories grammaticales spéciales dans un automate de phrase

Le reste de cette section est consacré à une présentation plus en détail de ces différentes catégories grammaticales, qui ne sont pas déclarées dans la description *lingdef* puisque indépendantes de la langue.

2.4.1 Le mot vide

Les transitions étiquetées par le mot vide sont souvent utilisées dans les grammaires locales car elles permettent d'améliorer leur lisibilité. Le mot vide est représenté dans les graphes par le symbole <E>. Nous avons ainsi pris le parti de représenter ce symbole particulier par un masque lexical.

Évidemment, ce type de masque nécessite des traitements spécifiques pendant l'analyse, puisque, contrairement aux autres transitions, les transitions étiquetées par <E> peuvent être franchies sans condition à chaque étape du calcul et sans consommer de symbole en entrée. Dans la pratique, nous appliquons un algorithme de suppression des transitions étiquetées par le mot vide pendant la phase d'optimisation des grammaires, en préambule à leur confrontation avec des textes. Ceci nous permet de ne pas avoir à traiter ce type de symbole durant l'analyse.

2.4.2 Les chiffres et les symboles de ponctuation

Les catégories grammaticales prédéfinies *number* (ayant NB pour nom alternatif) et *punc* (ayant PNC pour nom alternatif) permettent de représenter respectivement les tokens numériques et les différents symboles de ponctuation. Les unités lexicales de ce type sont déjà identifiées en tant que telles durant la phase de découpage en tokens (cf. section 3.2.1) du texte qui a lieu en amont de la phase d'étiquetage morpho-syntaxique. Ainsi, lors de la construction de l'automate du texte, les masques lexicaux représentant ces deux types de tokens sont automatiquement étiquetés avec la catégorie grammaticale qui leur correspond.

2.4.3 Catégorie LEX

La catégorie LEX n'est pas réellement indispensable pour la représentation du texte ou des grammaires. Nous l'avons rajoutée à des fins d'optimisation pour le traitement des grammaires qui contiennent des transitions étiquetées

par des mots sans étiquette grammaticale. En effet, lorsque nous confrontons les grammaires avec l'automate du texte, ce type de symbole consistant en un mot sans étiquette, concorde avec l'ensemble des analyses lexicales de ce mot obtenues lors de la phase d'étiquetage morpho-syntaxique par consultation des dictionnaires. Ainsi, si nous prenons par exemple la grammaire de la figure 2.5, cette grammaire reconnaît la séquence de mots *le président de la république* indépendamment de l'analyse lexicale de ses éléments. La recherche de séquences concordant avec cette grammaire sur l'automate du texte représentant cette même séquence (représenté dans la figure 2.6) nous donnerait 40 $(2 * 2 * 5 * 2)$ occurrences pour ce seul segment du fait de l'ambiguïté d'étiquetage morpho-syntaxique.

FIG. 2.5 – Grammaire avec des mots sans étiquettes

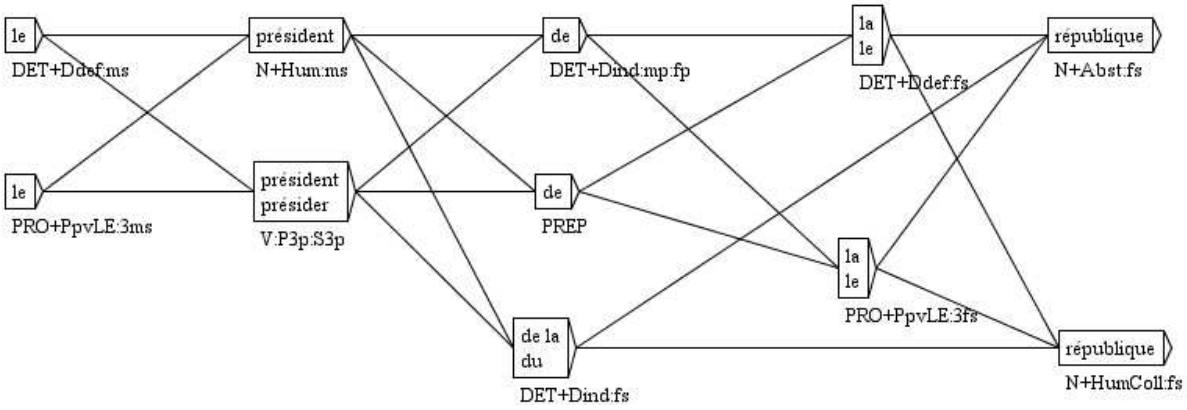

FIG. 2.6 – Ambiguïté d'étiquetage morpho-syntaxique

En effet, le résultat de la recherche de séquences concordant avec une grammaire sur l'automate du texte consiste en un ensemble de chemins dans cet automate et non en un ensemble de segments du texte comme c'est le cas avec une représentation linéaire du texte. Or, le nombre d'analyses lexicales pour un segment donné — c'est-à-dire le nombre de chemins dans l'automate du texte entre deux états — est typiquement exponentiel en la taille de ce

segment (typiquement, après l'application des dictionnaires DELA des mots simples et composés sur un texte français, nous obtenons plus de 2 analyses par mot en moyenne). Ainsi, la présence de mot sans étiquette grammaticale dans les grammaires est susceptible d'alourdir les traitements lors de leur application sur les textes et de retourner un nombre élevé d'analyses dont les éléments qui les distinguent ne sont pas nécessairement pertinents pour le grammairien. On peut en effet raisonnablement penser que lorsqu'il utilise ce type de symbole, le grammairien est plutôt intéressé par la présence d'un mot dans un certain contexte, que par ses possibles étiquettes grammaticales.

Pour toutes ces raisons, nous avons ajouté une pseudo catégorie grammaticale LEX pour représenter les symboles sans étiquette grammaticale. Lors de la traduction d'une grammaire en automate lexical, nous étiquetons systématiquement tous les symboles de ce type par la catégorie grammaticale LEX, comme le montre la figure 2.7 qui présente la grammaire de la figure 2.5 sous la forme d'un automate lexical.

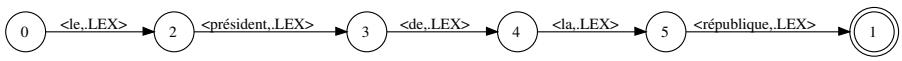

FIG. 2.7 – Automate lexical correspondant à la grammaire de la figure 2.5

Parallèlement, lors de la construction de l'automate du texte, nous ajoutons pour chaque mot du texte une transition de catégorie LEX en parallèle aux transitions correspondant aux analyses issues de la consultation des dictionnaires. La figure 2.8 présente l'automate que nous obtenons pour la séquence *Le président de la république.*

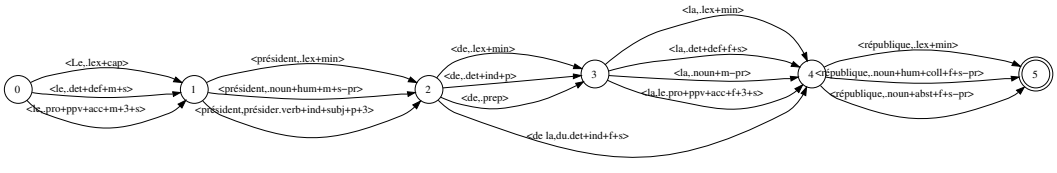

FIG. 2.8 – Automate lexical avec catégorie LEX

Grâce à l'ajout de la catégorie LEX et à ces pré-traitements sur le texte et les grammaires, nous pouvons traiter les mots sans étiquette dans les grammaires exactement de la même manière que n'importe quel autre masque lexical. De plus, de tels symboles ne concordent plus qu'avec une seule analyse pour chaque occurrence du mot dans le texte, ce qui permet d'une part d'accélérer les traitements et d'autre part de réduire considérablement le bruit produit dans les résultats d'analyse.

Notons qu'avec cette approche nous modifions sensiblement l'interprétation que nous donnons à l'automate du texte. En effet, avec l'ajout de cette pseudo catégorie LEX, nous utilisons les chemins parallèles dans l'automate pour représenter autre chose que des ambiguïtés lexicales, ce qui complique légèrement le modèle.

2.4.4 Les mots inconnus

Il est également possible de faire de la recherche de mots inconnus des dictionnaires dans une grammaire locale à l'aide du symbole <!DIC>. La reconnaissance de telles occurrences demande un traitement particulier puisqu'elle consiste à rechercher les mots qui n'ont reçu aucune étiquette grammaticale à l'issue de l'étiquetage morpho-syntaxique. Dans le cas d'une recherche sur l'automate du texte, l'opération consisterait à chercher les états dans l'automate qui ont une unique transition sortante de catégorie LEX. Nous avons préféré traiter la reconnaissance des mots inconnus de la même manière que la reconnaissance de n'importe quel autre motif. A cet effet, lors de la construction de l'automate du texte par consultation de dictionnaires, nous ajoutons automatiquement une transition étiquetée par la catégorie *unknown* en parallèle à la transition de catégorie LEX, pour tous les mots du texte qui n'ont pas d'entrée dans le dictionnaire. Ainsi la reconnaissance d'un mot inconnu consiste simplement à retrouver les mots étiquetés par la catégorie *unknown* comme pour la reconnaissance de n'importe quelle autre catégorie grammaticale.

2.4.5 Récapitulatif des correspondances avec les symboles META d'INTEX

Le tableau suivant présente les correspondances permettant de représenter les symboles META utilisés dans les grammaires locales par des masques lexicaux. Lors de la traduction des grammaires en automates lexicaux, nous détectons automatiquement ces types de symboles et nous les transformons dans le masque qui leur correspond. Le cas du symbole <DIC> est un peu particulier, puisque ce symbole, qui concorde avec l'ensemble des mots qui ont reçu une analyse lexicale lors de la consultation des dictionnaires, ne peut pas être représenté par un simple masque lexical. Cependant cet ensemble est équivalent à l'union de toutes les catégories grammaticales définies dans la description du jeu d'étiquettes. Ainsi, lors de la traduction des grammaires locales, nous remplaçons les transitions étiquetées par <DIC> par un ensemble de transitions parallèles étiquetées par des masques lexicaux pour chacune de ces catégories grammaticales.

<TOKEN>	<.>
<MOT>	<LEX>
<MIN>	<LEX+min>
<MAJ>	<LEX+maj>
<PRE>	<LEX+cap>
<PNC>	<PNC> (nom alternatif pour la catégorie punc)
<NB>	<NB> (nom alternatif pour la catégorie *number*)
<DIC>	<V>+<N>+<A>+... disjonction de toutes les catégories grammaticales définies dans le lingdef
<!DIC>	<unknown>

FIG. 2.9 – Correspondant entre symboles META et masques lexicaux

2.5 Opérations ensemblistes sur les masques lexicaux

Grâce à notre description formelle du jeu d'étiquettes, nous obtenons une représentation homogène et structurée des étiquettes grammaticales dans laquelle chaque mot, selon sa catégorie grammaticale, admet un ensemble déterminé d'attributs qui peuvent prendre une valeur parmi des codes bien définis. Nous rapprochons ainsi notre description de travaux récents sur la représentation des ressources linguistiques [EAGLES, 1996, Role, 1999, Francopoulo, 2003] qui intègrent directement la notion d'attribut et de catégories de données dans leur modèle.

Dans ce cadre, nous pouvons représenter simplement un masque lexical par une structure de données à plusieurs champs identifiant ses éventuelles formes fléchie et canonique, sa partie du discours et les traits qui lui sont attribués ; ces traits sont stockés dans un tableau d'attributs dont la taille et l'interprétation de ses éléments sont définies dans notre description en fonction de la partie du discours. Ainsi, d'après notre description de la figure 2.3 (page 20), nous associons à un masque lexical de la catégorie des noms un tableau de six attributs : les trois premiers sont de type énumération et caractérisent sa sous-catégorie sémantique, son genre et son nombre ; les trois derniers sont de type booléen et spécifient s'il s'agit d'un nom propre, d'un nom composé et d'un nom collectif.

Nous représentons chacun de ces attributs morpho-syntaxiques par un champ de bits de manière à pouvoir représenter une disjonction de valeurs atomiques. La valeur spéciale qui indique que tel attribut n'est pas spécifié dans le masque (et donc qu'il peut prendre n'importe quelle valeur) est représentée par un champ où tous les bits sont positionnés à 1.

En ce qui concerne la représentation des formes fléchies et canoniques, les attributs de ce type peuvent prendre comme valeur une disjonction de chaînes de caractères (comme par exemple dans le masque <rouge|noir.adj> qui décrit l'ensemble des formes fléchies des adjectifs *rouge* et *noir*) ou bien une négation d'un ensemble de chaînes de caractères (par exemple dans <!blanc!bleu.adj> qui décrit l'ensemble des adjectifs excepté ceux ayant

blanc ou *bleu* pour lemme) ; ces attributs ne peuvent donc pas être représentés par un champ de bits de taille finie. Nous les représentons par une liste chaînée de chaînes de caractères triées par ordre lexicographique à laquelle est associée un booléen qui spécifie si l'ensemble de ces chaînes est nié ou non.

A partir d'une telle représentation, nous sommes parvenu à implémenter des opérations ensemblistes sur les masques lexicaux, telles que le calcul de l'intersection ou de la différence de deux masques. Nous présentons dans la suite de cette section les algorithmes qui implémentent ces deux opérations, qui se sont avérées être nécessaires pour l'implémentation d'algorithmes de plus haut niveau, opérant sur les automates lexicaux.

2.5.1 Représentation formelle

Formellement, nous pouvons définir un masque lexical comme une conjonction de n prédicats, le nombre n étant dépendant de la catégorie grammaticale du masque. Chaque prédicat sélectionne un sous-ensemble du lexique en fonction de la valeur d'un trait ; ce trait peut porter sur la forme fléchie, le lemme, la catégorie grammaticale, et les traits morpho-syntaxiques définis dans la description du jeu d'étiquettes.

Par exemple, d'après notre description pour le français de la figure 2.3, le masque <masque.noun+conc+abst+m> peut être décrit comme la conjonction de 9 prédicats, dont 4 explicites :

- le premier sélectionne l'ensemble des mots du lexique ayant *masque* pour forme canonique ;
- le second sélectionne l'ensemble des substantifs ;
- le troisième sélectionne les noms appartenant à la sous-catégorie des noms concrets ou abstraits ;
- et le dernier prédicat sélectionne les mots de genre masculin.

A ces 4 prédicats s'ajoutent cinq prédicats implicites correspondant aux traits qui ne sont pas spécifiés dans le masque, c'est-à-dire les prédicats portant sur la forme fléchie et le nombre des entrées sélectionnées ainsi que les attributs

spécifiant s'il s'agit d'un nom propre, d'un nom composé et d'un nom collectif. Comme ils ne sont pas spécifiés dans le masque, ces cinq derniers prédicats sélectionnent la totalité du lexique.

Ainsi, l'ensemble des mots étiquetés décrits par un masque m correspond à l'ensemble des mots qui sont sélectionnés par tous les prédicats qui le composent, ce qui équivaut à l'intersection des ensembles décrits par ces n prédicats. Par la suite, on assumera, sans perte de généralité, que n est constant et indépendant de la catégorie grammaticale. Étant donné un masque lexical m, on notera m_i le i^{eme} prédicat qui compose ce masque. On a donc, pour tout masque m, l'égalité :

$$m = m_1 \wedge m_2 \wedge ... \wedge m_n$$

2.5.2 Calcul de l'intersection

Le calcul de l'intersection de deux masques lexicaux s'effectue très simplement. L'opération consiste à calculer le masque lexical qui combine les informations spécifiées dans les deux masques si elles sont compatibles ; dans le cas contraire, l'ensemble vide est retourné. Ce calcul s'apparente ainsi à l'opération d'unification (cf. section 4.3.1) qui est à la base de nombreux formalismes linguistiques [Bresnan, 1982, Gazdar *et al.*, 1985, Pollard et Sag, 1994]. Seulement, nous traitons ici des structures de traits aplaties puisque la valeur d'un trait ne peut pas être elle-même une structure de traits. Nous présentons le pseudo-code de cette opération dans l'algorithme 1.

Nous commençons par vérifier que les masques a et b sont de même catégorie grammaticale ; dans le cas contraire les deux masques sont clairement disjoints et l'on retourne l'ensemble vide (ligne 2). Ensuite, nous parcourons les attributs de a et b en parallèle et nous assignons à l'attribut correspondant de m, l'intersection de ces deux attributs. Si cette intersection est vide, alors les masques a et b sont disjoints et le masque vide est retourné (ligne 8). A la sortie de la boucle nous retournons le masque m ainsi construit.

Les traits morpho-syntaxiques (de type booléen ou énumération) sont représentés par des champs de bits. Ainsi, l'intersection de deux attributs de ce

entrée: a et b, deux masques lexicaux
sortie: m un masque lexical tel que l'ensemble des entrées lexicales décrites dans m est exactement l'ensemble des entrées appartenant aux ensembles décrits par a et b ; si cette intersection est vide, retourne $\emptyset$.

1: **si** $cat(a) \neq cat(b)$ **alors**
2: **retourner** $\emptyset$
3: **fin si**
4: $cat(m) \leftarrow cat(a)$
5: **pour tout** i tel que $1 \leq i \leq n$ **faire**
6: $m_i \leftarrow a_i \wedge b_i$
7: **si** $m_i = \emptyset$ **alors**
8: **retourner** $\emptyset$
9: **fin si**
10: **fin pour**
11: **retourner** m

Algorithme 1: Calcul de l'intersection de deux masques lexicaux

type se fait par un simple ET bit à bit sur ces deux champs. L'opération a donc un coût constant.

En ce qui concerne les attributs représentant les formes fléchies et canoniques, rappelons qu'ils sont représentés par une liste triée de formes à laquelle est associé un booléen spécifiant si ces formes sont niées ou non. L'implémentation du calcul de l'intersection de deux traits de ce type dépend ainsi de la valeur de ces booléens :

– si les deux ensembles sont positifs, le calcul consiste simplement à construire l'intersection des deux ensembles :
bleu|rouge $\wedge$ rouge|vert = rouge
– si un ensemble est positif, et l'autre négatif, l'opération consiste à retirer du premier ensemble les chaînes qui sont présentes dans le second. Le résultat est un ensemble positif :
bleu|rouge|vert $\wedge$!bleu !vert = rouge
– si les deux traits représentent des formes niées, alors l'opération consiste à calculer l'union de ces deux ensembles. Le résultat est un ensemble négatif :

!bleu !vert ∧ !jaune !vert = !bleu !jaune !vert

Ainsi, quelle que soit la forme des deux attributs, le calcul de l'intersection s'implémente par un simple parcours unidirectionnel des deux listes en parallèle, puisque celles-ci ont été préalablement triées. Nous construisons durant ce parcours l'ensemble résultat en choisissant à chaque étape quelles sont les formes considérées qui doivent être incluses dans cet ensemble. Le coût de cette opération est donc linéaire en la somme des tailles de ces deux ensembles.

Les lignes qui suivent présentent quelques exemples d'intersection de deux masques lexicaux :

(a) <!noir.adj> ∧ <!rouge.adj+m+s> = <!rouge !noir.adj+m+s>

(b) <!noir.adj+f> ∧ <rouge.adj+s> = <rouge.adj+f+s>

(c) <noun+m> ∧ <noun+f> = ∅

(d) <verb+P+S> ∧ <verb+1+3+s> = <verb+P+S+1+3+s>

(e) <verb+P> ∧ <verb+m> = <verb+P+m>

A la ligne (e), le masque <verb+P+m>, résultat de l'opération ne concorde en pratique avec aucune entrée lexicale, puisque le trait de temps *présent* et celui de genre *masculin* ne sont pas compatibles : le trait de genre n'est positionné pour les verbes dans nos dictionnaires que pour les formes au participe passé. Dans une version antérieure de notre implémentation (présentée dans [Blanc et Dister, 2004]), nous détections les masques de ce type et nous les supprimions automatiquement des grammaires. Cette fonctionnalité nécessitait d'enrichir la description du jeu d'étiquettes d'une nouvelle section décrivant, pour chaque catégorie grammaticale, les combinaisons de traits compatibles entre elles. Cependant, notre implémentation de l'époque ne permettait pas de représenter des masques lexicaux ayant plusieurs valeurs possibles pour un même trait, comme dans le masque <verb+P+S+1+3+s> où le trait de temps peut prendre les valeur P ou S (présent de l'indicatif ou du subjonctif) et où l'attribut de personne peut avoir les valeurs 1 ou 3. Un tel ensemble devait être représenté par l'union des 4 masques lexicaux : <verb+P+1+s>, <verb+P+3+s>, <verb+S+1+s> et <verb+S+3+s>. La détection de la présence de traits incompatibles dans un masque lexical avec

notre nouvelle représentation nécessiterait d'étudier toutes les combinaisons de traits séparément de manière à détecter les combinaisons invalides. Cette opération nous a semblé trop coûteuse pour un bénéfice qui n'est pas considérable. Nous avons donc préféré abandonner cette fonctionnalité au profit d'une représentation des masques lexicaux permettant de factoriser un grand nombre d'ambiguïtés morpho-syntaxiques.

2.5.3 Calcul de la différence

Nous avons également implémenté le calcul de la différence de deux masques lexicaux, qui est présenté en pseudo-code dans l'algorithme 2. Le résultat de cette opération a la forme générale d'une union de masques lexicaux deux à deux disjoints.

entrée: a et b, deux masques lexicaux
sortie: S, une union de masques lexicaux deux à deux disjoints tels que l'ensemble décrit par S est exactement l'ensemble décrit par a privé de l'ensemble décrit par b.

1: **si** $cat(a) \neq cat(b)$ **alors**
2: **retourner** $\{a\}$
3: **fin si**
4: $S \leftarrow \emptyset$
5: $m \leftarrow a$
6: **pour tout** i, $1 \leq i \leq n$ **faire**
7: $m_i \leftarrow a_i \wedge \overline{b_i}$
8: **si** $m_i \neq \emptyset$ **alors**
9: $S \leftarrow S \cup m$
10: **fin si**
11: $m_i \leftarrow a_i \wedge b_i$
12: **fin pour**
13: **retourner** S

Algorithme 2: Calcul de la différence de deux masques lexicaux

Si la catégorie grammaticale de a est différente de celle de b, alors les deux ensembles qu'ils décrivent sont disjoints et nous retournons la masque a

(ligne 2). Sinon, on initialise le masque m avec la valeur de a (ligne 5). Ce masque m servira à construire de façon incrémentale les masques qui constitueront l'ensemble S résultat. On parcourt ensuite l'ensemble des attributs de a et b ; à la i^{eme} itération, on assigne à m_i la valeur représentant l'ensemble décrit par a_i privé de l'ensemble décrit par b_i (ligne 7). Si m_i est non vide, alors on rajoute m à l'ensemble S résultat. Puis, on assigne à m_i, la valeur représentant l'intersection des deux ensembles décrits par a_i et b_i (ligne 11) et on continue sur la prochaine itération.

Ainsi, l'ensemble S retourné par notre algorithme peut être traduit en notation ensembliste par la disjonction suivante :

$$\begin{aligned} S = \quad & ((a_1 \wedge \overline{b_1}) \wedge a_2 \wedge \cdots \wedge a_n) \\ & \vee ((a_1 \wedge b_1) \wedge (a_2 \wedge \overline{b_2}) \wedge \cdots \wedge a_n) \\ & \vdots \\ & \vee ((a_1 \wedge b_1) \wedge (a_2 \wedge b_2) \wedge \cdots \wedge (a_n \wedge \overline{b_n})) \end{aligned}$$

où chaque élément de la disjonction (chaque ligne de l'équation) correspond à un masque de l'ensemble S. Ces masques sont bien disjoints deux à deux.

preuve

Pour tout i, j tel que $1 \leq i < j \leq n$, on a les égalités :

$$S_i = (a_1 \wedge b_1) \wedge \cdots \wedge (a_i \wedge \overline{b_i}) \wedge ... \wedge a_j \wedge \cdots \wedge a_n$$
$$S_j = (a_1 \wedge b_1) \wedge \cdots \wedge (a_i \wedge b_i) \wedge ... \wedge (a_j \wedge \overline{b_j}) \wedge \cdots \wedge a_n$$

et donc

$$S_i \subseteq a_i \wedge \overline{b_i b_i} \qquad\qquad S_j \subseteq a_i \wedge b_i \subseteq b_i$$

et donc

$$m_i \wedge m_j = \emptyset$$

D'autre part, nous montrons dans les équations suivantes, que l'ensemble S ainsi calculé décrit bien l'ensemble des mots étiquetés décrits par le masque

a mais non par le masque b, ce qui termine la preuve de la correction de notre algorithme.

preuve

L'ensemble S peut être réécrit sous la forme suivante :

$$\begin{aligned} S = \quad & (a_1 \wedge a_2 \wedge \cdots \wedge a_n) \wedge \overline{b_1} \\ & = \vee\, (a_1 \wedge a_2 \wedge \cdots \wedge a_n) \wedge b_1 \wedge \overline{b_2} \\ & \quad \vdots \\ & = \vee\, (a_1 \wedge a_2 \wedge \cdots \wedge a_n) \wedge b_1 \wedge b_2 \wedge \cdots \wedge \overline{b_n} \end{aligned}$$

En factorisant par la conjonction $(a_1 \wedge a_2 \wedge \cdots \wedge a_n)$ nous obtenons l'égalité :

$$\begin{aligned} S &= (a_1 \wedge a_2 \wedge \cdots \wedge a_n) \wedge (\overline{b_1} \vee (b_1 \wedge \overline{b_2}) \vee ... \vee (b_1 \wedge \cdots \wedge b_{n-1} \wedge \overline{b_n})) \\ &= a \wedge (\overline{b_1} \vee (b_1 \wedge \overline{b_2}) \vee ... \vee (b_1 \wedge \cdots \wedge b_{n-1} \wedge \overline{b_n})) \\ &= a \wedge (\overline{b_1} \vee (b_1 \wedge (\overline{b_2} \vee (b_2 \wedge \cdots b_n)))) \end{aligned}$$

Puisque pour tous prédicats x et y, on a l'égalité :

$$\overline{x} \vee (x \wedge y) = \overline{x} \vee y$$

nous pouvons réécrire S sous la forme suivante :

$$\begin{aligned} S &= a \wedge (\overline{b_1} \vee \overline{b_2} \vee \cdots \vee \overline{b_n}) \\ &= a \wedge (\overline{b_1 \wedge b_2 \wedge \cdots \wedge b_n}) \\ &= a \wedge \overline{b} \end{aligned}$$

Les lignes suivantes présentent quelques exemples du calcul de la différence d'un masque par rapport à un autre :

(1) <!noir.noun> - <rouge.noun> = < !noir !rouge.noun>

(2) <!noir.noun> - <!rouge.noun> = <rouge.noun>

(3) <verb+P> - <verb+1+s> = <verb+P+2> ∨ <verb+P+3> ∨ <verb+P+1+p>
(4) <verb+P> - <verb+1+s> = <verb+P+p> ∨ <verb+P+2+s> ∨ <verb+P+3+s>

On remarque que les exemples (3) et (4) donnent deux résultats différents pour la même opération. Cette différence est due au fait que le contenu de l'ensemble S construit dépend de l'ordre choisi par notre algorithme pour parcourir les attributs. Dans la ligne (3), le résultat a été construit en traitant d'abord l'attribut de personne, puis l'attribut de nombre. En (4), l'ordre a été inversé. Notons qui si ces résultats sont différents, les ensembles de mots étiquetés qu'ils décrivent sont bien identiques : l'ensemble des verbes conjugués au présent de l'indicatif, mais non à la première personne du singulier.

2.6 Opérations sur les automates lexicaux

Grâce à cette représentation structurée des masques lexicaux permettant d'implémenter des opérations ensemblistes sur ces objets, nous sommes maintenant en mesure d'adapter les algorithmes classiques sur les automates finis (tels que définis dans [Hopcroft et Ullman, 1979] par exemple) de manière à ce qu'ils aient un comportement correct sur les automates lexicaux. Les modifications apportées à ces algorithmes consistent essentiellement à découper les transitions afin que les ensembles décrits dans leurs étiquettes soient deux à deux soit disjoints, soit égaux. Nous procédons à ce découpage sur les transitions sortantes des états considérés à chaque étape du calcul de manière à ce que deux masques distincts susceptibles d'être comparés ont toujours une intersection vide. Cette propriété est suffisante à la correction de la plupart des algorithmes et nous avons pu ainsi implémenter le calcul de la déterminisation, la minimisation ou de l'intersection pour les automates lexicaux. La figure 2.10, par exemple, présente l'automate de la figure 2.2 (page 15) correctement déterminisé.

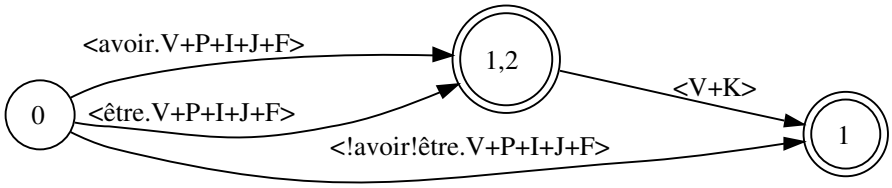

FIG. 2.10 – Automate lexical déterministe

Le calcul de la complémentation d'un automate lexical est différent puisqu'il consiste à inverser la terminalité des états de l'automate après l'avoir rendu complet. Pour ce faire nous rajoutons, pour chaque état, de nouvelles transitions sortantes dirigées vers un état puits et étiquetées par l'ensemble du lexique qui n'est pas décrit par les transitions déjà existantes, ensemble qu'il nous est maintenant possible de calculer.

Notons que le calcul de l'union ou de la concaténation de deux automates lexicaux ou encore le calcul de l'étoile de Kleen d'un automate lexical ne nécessitent pas de manipulation particulière sur les transitions.

Dans cette section, après avoir donné une définition formelle des automates que nous manipulons, nous présentons notre algorithme de découpage des transitions, qui prend en entrée un ensemble de transitions et produit en sortie un ensemble équivalent tel que les masques qui étiquettent ces transitions soient deux à deux disjoints ou égaux. Nous illustrons ensuite l'intérêt d'un tel découpage en détaillant notre algorithme qui implémente la déterminisation d'un automate lexical. Nous avons implémenté sur le même principe l'opération d'intersection de deux automates lexicaux.

2.6.1 Définition formelle

Étant donné un jeu d'étiquettes, nous définissons formellement un automate lexical non déterministe comme un quintuplet $A = (Lex, Q, I, F, \Delta)$ où :

– Lex est l'ensemble des masques lexicaux définis sur le jeu d'étiquettes

utilisé ;
– Q est un ensemble fini d'état ;
– $I \subseteq Q$ est l'ensemble des états initiaux ;
– $F \subseteq Q$ est l'ensemble des états finaux ;
– $\Delta \subseteq Q \times Lex \times Q$ est un ensemble de transitions lexicales.

Une transition lexicale est définie comme un triplet $t = (q_o, m, q_d)$ où

– q_o est l'état d'origine,
– m est le masque lexical qui étiquette cette transition et
– q_d l'état de destination.

Étant donnée une transition lexicale t, on notera $orig(t)$, $label(t)$ et $dest(t)$ ces trois éléments respectifs.

De la même manière, nous définissons un automate lexical déterministe comme un quintuplet $A = (Lex, Q, i, F, \delta)$ où :

– Lex est l'ensemble des masques lexicaux ;
– Q est un ensemble fini d'état ;
– $i \in Q$ est l'état initial ;
– $F \subseteq Q$ est l'ensemble des états finaux ;
– $\delta : Q \times Lex \mapsto Q$ est la fonction de transition ;
– Pour tout $q \in Q$ et pour tout $m_1, m_2 \in Lex$, si $\delta(q, m_1)$ et $\delta(q, m_2)$ sont définis, alors soit $m_1 \wedge m_2 = \emptyset$ soit $m_1 = m_2$.

2.6.2 Découpage d'un ensemble de transitions

Le découpage d'une liste de transitions est présenté en pseudo-code dans l'algorithme 3. Cet algorithme prend en entrée un ensemble de transitions lexicales T_1 et produit en sortie un nouvel ensemble T_2 équivalent tel que les masques lexicaux qui étiquettent les transitions de T_2 soient deux à deux disjoints ou égaux.

A chaque itération, l'algorithme extrait une transition t_1 de l'ensemble T_1

entrée: T_1 un ensemble de transitions étiquetées par des masques lexicaux
sortie: T_2, un ensemble de transitions équivalent à T_1 tel que les masques qui étiquettent les transitions de T_2 sont deux à deux disjoints ou égaux.

```
1:  T_2 ← ∅
2:  tant que T_1 ≠ ∅ faire
3:    t_1 ← DEQUEUE(T_1)
4:    dejapresent ← faux
5:    pour tout t_2 dans T_2 faire
6:      i ← label(t_1) ∧ label(t_2)
7:      si i ≠ ∅ alors
8:        pour tout masque m dans label(t_1) − i faire
9:          ENQUEUE(T_1, (orig(t_1), m, dest(t1)))
10:       fin pour
11:       label(t_1) ← i
12:       pour tout masque m dans label(t_2) − i faire
13:         ENQUEUE(T_2, (orig(t_2), m, dest(t_2)))
14:       fin pour
15:       label(t_2) ← i
16:       si t_1 = t_2 alors
17:         dejapresent ← vrai
18:       fin si
19:     fin si
20:   fin pour
21:   si dejapresent = faux alors
22:     ENQUEUE(T_2, t_1)
23:   fin si
24: fin tant que
```

Algorithme 3: Découpage d'une liste de transitions

(ligne 3), pour l'introduire dans l'ensemble T_2 (ligne 22) en procédant à un éventuel découpage des transitions de manière à ce qu'à chaque étape les masques qui étiquettent les transitions de cet ensemble soient deux à deux disjoints ou égaux. Pour ce faire, on parcourt toutes les transitions déjà présentes dans T_2. Si l'on rencontre une transition t_2 telle que l'intersection i des masques lexicaux qui étiquettent les deux transitions t_1 et t_2 est non vide

(ligne 7), on procède alors à un découpage de ces deux transitions : on réintroduit dans l'ensemble T_1 les transitions étiquetées par $label(t_1) - i$ (lignes 8 et 9) de manière à les traiter lors d'une prochaine itération de la boucle principale et on assigne i pour nouvelle étiquette de t_1 (ligne 10) ; on découpe la transition t_2 de la même manière en mettant à jour l'ensemble T_2 (lignes 12 à 15). Une fois parcouru T_2, on introduit finalement la nouvelle transition t_1 dans cet ensemble à la condition qu'elle n'y soit pas déjà présente (lignes 20 à 22). L'algorithme se termine lorsque l'ensemble T_1 est épuisé.

Sans faire le calcul exact de la complexité de l'algorithme, les deux boucles imbriquées qui parcourent les deux ensembles T_1 et T_2 suggèrent que cette complexité est quadratique dans le meilleur des cas. De plus, chaque insertion d'une nouvelle transition dans T_2 peut causer des découpages de transitions et donc agrandir la taille de ces deux ensembles. Il est cependant possible d'accélérer le calcul en procédant à certaines optimisations que nous avons implémentées mais qui n'ont pas été reportées dans le pseudo-code pour des raisons de lisibilité. Ainsi, lors du parcours des transitions de l'ensemble T_2 pour y insérer une nouvelle transition t_1, la première fois que l'étiquette de t_1 intersecte l'étiquette d'une transition t_2, nous conservons le résultat du calcul de la différence $label(t_2) - i$ calculée à la ligne 12, de manière à ne pas avoir à le recalculer lors des prochaines itérations. En effet, une fois le premier découpage effectué, nous continuons l'itération de l'ensemble T_2 avec une nouvelle transition t_1 ayant i pour étiquette (ligne 11). Ainsi, si nous rencontrons une autre transition t'_2 qui intersecte cette nouvelle transition t_1, nous savons que l'étiquette de t'_2 est nécessairement identique à celle de la première transition t_2 qui a déclenché le premier découpage, puisque nous savons que si deux étiquettes de l'ensemble T_2 ont une intersection non vide, alors ces étiquettes sont égales. Nous pouvons donc directement réutiliser le résultat du premier découpage pour découper la transition t'_2.

De la même manière, nous pouvons déduire de cette propriété que le découpage d'une transition t_1 n'arrive au plus qu'une fois lors de son insertion dans l'ensemble T_2. En effet, nous savons que les fois suivantes où l'étiquette de t_1 intersecte l'étiquette d'une nouvelle transition t'_2, le masque i calculé à la ligne 6 est égal à la nouvelle étiquette de t_1 (qui lui a été assignée ligne 11 lors du premier découpage) ; ainsi les lignes 8 à 11 de notre algorithme n'ont un réel effet que lors du premier découpage.

Nous avons ainsi pu accélérer sensiblement notre algorithme en divisant la boucle qui itère sur les transitions de T_2 en deux parties. La première partie s'interrompt à la rencontre de la première transition t_2 dont l'étiquette intersecte celle de t_1 (si une telle transition n'est pas présente alors on ajoute directement t_1 dans T_2 en ne procédant à aucun découpage). Si une telle transition existe, alors nous calculons une unique fois le découpage des deux transitions t_1 et t_2, puis nous rentrons dans une seconde boucle qui parcourt la fin de l'ensemble T_2 en remplaçant chaque transition t'_2 dont l'étiquette est égale à celle de t_2 selon le même modèle déjà calculé.

2.6.3 Algorithme de déterminisation

Nous donnons dans la figure 4 l'algorithme qui calcule la déterminisation d'un automate lexical. Notre algorithme prend en entrée un automate lexical non déterministe A_1 et produit en sortie un automate lexical déterministe A_2 qui reconnaît le même langage que A_1. L'algorithme est basé sur l'algorithme classique de construction de l'automate des sous-ensembles.

La seule modification apportée par rapport à l'algorithme original est à la ligne 10 où nous procédons à un découpage de toutes les transitions sortantes des états du sous-ensemble q_2 considéré, avant de calculer les transitions sortantes pour cet état dans l'automate A_2 déterministe (lignes 11 à 16). Ce découpage préalable permet de s'assurer que si deux transitions sortantes sont étiquetées par des masques distincts, alors l'intersection des ensembles décrits par ces deux masques est vide. Cette propriété suffit à la correction de notre algorithme.

Toutes ces opérations supplémentaires qui peuvent être effectuées efficacement grâce à notre représentation appropriée des masques lexicaux ont néanmoins un impact négatif sur l'efficacité générale de l'algorithme. En effet, le nombre de comparaisons et de découpages des transitions qui sont effectués dépend de la structure des automates en entrée et peut être, dans le pire des cas, exponentiel par rapport au nombre total de transitions.

Cependant, ce surcoût concerne uniquement les algorithmes de construction et d'optimisation des grammaires qui a lieu en amont de leur utilisation pour

entrée: $A_1 = (Lex, Q_1, I_1, F_1, \Delta_1)$, un automate lexical non déterministe
sortie: $A_2 = (Lex, Q_2, i_2, F_2, \delta_2)$, un automate lexical déterministe qui reconnaît le même langage que A_1

$F_2 \leftarrow \emptyset$
$i_2 \leftarrow \bigcup_{i \in I_1} \{i\}$
$Q \leftarrow \{i_2\}$
tant que $Q \neq \emptyset$ **faire**
  $q_2 \leftarrow$ DEQUEUE(Q)
  $Q_2 \leftarrow Q_2 \cup \{q_2\}$
  si il existe $q \in q_2$ tel que $q \in F_1$ **alors**
    $F_2 \leftarrow F_2 \cup \{q_2\}$
  fin si
  $T =$ DECOUPE_TRANS$(\{(q, m, q') \in \Delta_1, q \in q_2\})$
  pour tout masque m tel que $(q, m, q') \in T$ **faire**
    $\delta_2(q_2, m) \leftarrow \bigcup_{(q,m,q') \in T} \{q'\}$
    si $\delta_2(q_2, m)$ est un nouvel état **alors**
      ENQUEUE$(Q, \delta_2(q_2, m))$
    fin si
  fin pour
fin tant que

Algorithme 4: Déterminisation d'un automate lexical

l'analyse de textes. Le coût induit par ces opérations supplémentaires sur les transitions doit de ce fait être relativisé en considérant que l'utilisation d'automates correctement déterminisés apporte une nette amélioration des performances lors de leurs applications sur des corpus de texte (nous donnons des évaluations chiffrées des performances dans la section 2.8.5 qui traite du cas des automates lexicaux avec sorties).

2.7 Application à la levée d'ambiguïtés lexicales

Dans la suite de ce mémoire, nous aborderons des modèles grammaticaux plus puissants qui étendent le formalisme des automates lexicaux :

- les transducteurs lexicaux (cf. section 2.8), qui sont des automates lexicaux avec sorties,
- les grammaires WRTN (cf. section 3.3), qui sont composées d'un ensemble d'automates lexicaux récursifs et pondérés,
- et les grammaires DRTN (cf. section 4.4) qui augmentent le modèle des WRTN avec une composante d'unification.

Ces différentes extensions au modèle des automates lexicaux se sont avérées être adaptées pour différentes tâches en traitement automatique des langues, telles que l'annotation de texte, la reconnaissance d'entités, l'analyse de surface et l'analyse syntaxique profonde que nous évoquerons ensuite. Dans cette section, nous présentons le système de levée d'ambiguïtés lexicales ELAG (Elimination of Lexical Ambiguities by Grammars) dont le fonctionnement est basé exclusivement sur l'utilisation d'automates lexicaux. Ce système a initialement été conçu par Éric Laporte [Laporte et Monceaux, 1999] qui en a implémenté une première version compatible avec le logiciel INTEX. Le module était alors uniquement utilisable avec un jeu d'étiquettes qui contenait un sous-ensemble des codes syntaxiques et flexionnels utilisés dans le dictionnaire DELAF français [Courtois, 1990].

Notre implémentation des algorithmes pour la manipulation des automates et des masques lexicaux nous a permis de réimplémenter de façon triviale le système. De plus, le paramétrage de ces fonctionnalités par une description extérieure du jeu d'étiquettes nous a permis d'écrire ces algorithmes indépendamment du jeu d'étiquettes utilisé et donc d'utiliser ELAG pour traiter des textes dans d'autres langues que le français. Nous avons ainsi intégré ELAG au système Unitex, ce qui a permis à plusieurs équipes de recherche d'utiliser le module pour la désambiguïsation des textes dans de nombreuses langues, telles que le français [Blanc et Dister, 2004], le grec moderne [Blanc *et al.*, 2005], le portugais, l'espagnol [Cunqueiro, 2005], l'allemand et

le serbe.

Notre système est basé sur l'utilisation de dictionnaires électroniques à large couverture et sur des règles de désambiguïsation construites par des experts humains. Par ailleurs, notre approche vise à fournir le résultat de la désambiguïsation non pas sous la forme d'un texte linéaire totalement étiqueté, comme c'est classiquement le cas [Schmid, 1994, Brill, 1995], mais sous la forme d'un automate du texte partiellement désambiguïsé (cf. [Koskenniemi, 1990]).

2.7.1 Constitution d'une Grammaire ELAG

Une grammaire Elag a une syntaxe particulière. Elle est composée de deux parties que nous nommerons *partie si* et *partie alors* :

- La partie *si* est délimitée par deux symboles <!> et décrit le contexte dans lequel la grammaire va s'appliquer sur l'automate du texte. Ce motif est lui-même divisé en deux parties séparées par un symbole <!>.
- La partie *alors* est constituée de zéro, un ou plusieurs motifs délimités par des symboles <=> décrivant des contraintes morpho-syntaxiques qui doivent être satisfaites par les chemins de l'automate du texte qui concordent avec la partie *si* de la grammaire. Chaque contrainte est également divisée en deux parties délimitées par un symbole <=>.

Une grammaire de désambiguïsation doit être lue comme suit : si un chemin de l'automate du texte concorde avec la partie *si* de la grammaire, alors ce même chemin doit également concorder avec au moins une des contraintes spécifiée dans la partie *alors* ; dans le cas contraire, la séquence est dite rejetée par la grammaire et est supprimée de l'automate du texte.

Par exemple, la grammaire présentée dans la figure 2.11 considère la forme *se*, pronom personnel ; la partie *alors* de la grammaire décrit le contexte d'apparition de cette forme ainsi que les contraintes d'accord avec le verbe : soit le pronom est au singulier et il est suivi d'un verbe à la 3e personne du singulier (<V :3s>) ; soit le pronom est au pluriel et il est suivi d'un verbe à la 3e personne du pluriel (<V :3p>). Le verbe peut également être à l'infinitif (W) ou au gérondif (G). Entre le pronom *se* et le verbe, il est possible de

rencontrer un pronom préverbal soit objet (PRO+PpvLE, c'est-à-dire *le*, *la*, *les*), soit prépositionnel (PRO+PpvPR, c'est-à-dire *en* et *y*).

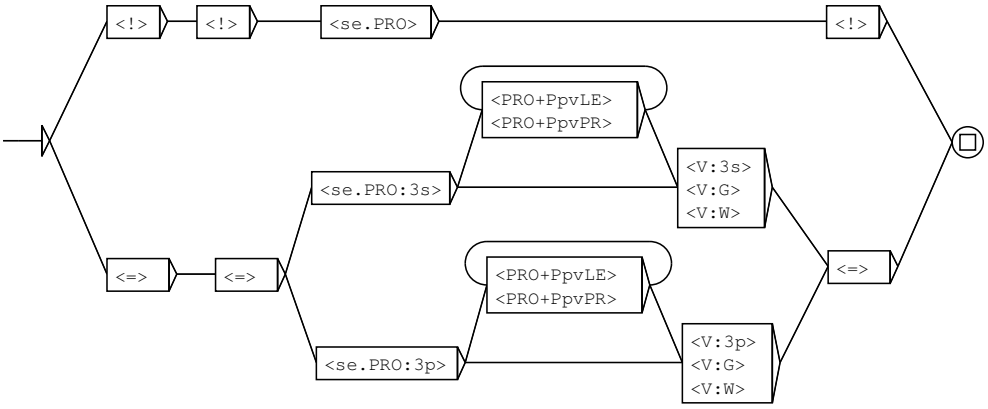

FIG. 2.11 – Contrainte d'accord grammatical entre le pronom se et le verbe

Nous montrons dans les figures 2.12 et 2.13 un exemple d'utilisation d'une grammaire ELAG. La figure 2.12 présente l'automate du texte obtenu après l'application des dictionnaires sur la phrase *Luc se le donne*. La figure 2.13 présente l'automate résultant de l'application de notre grammaire sur l'automate de phrase précédent.

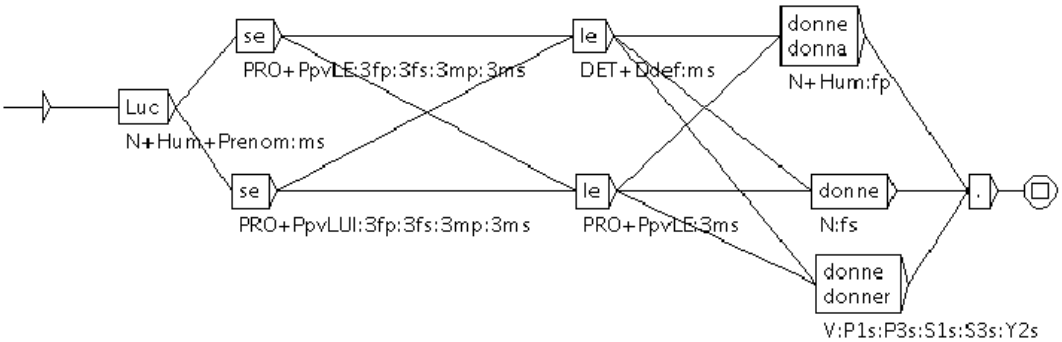

FIG. 2.12 – Automate de phrase

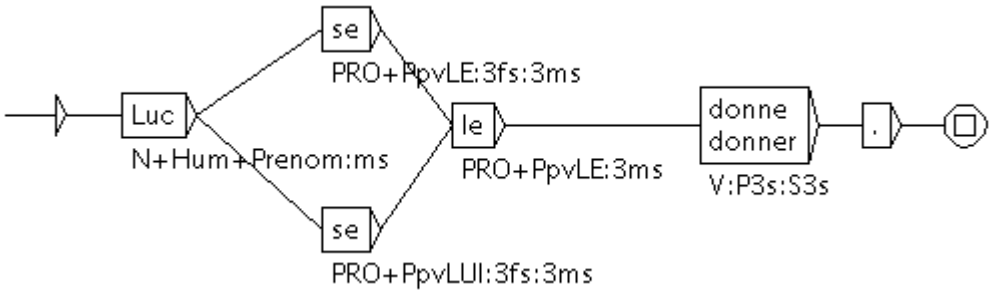

FIG. 2.13 – Automate de phrase partiellement désambiguïsé

Le point de synchronisation

Chaque partie *si* et *alors* d'une grammaire ELAG est divisée en deux zones par le symbole <!> et <=> respectivement. Le but de ce symbole délimiteur est de faire la synchronisation entre les motifs décrits dans les différentes parties de la grammaire. En particulier, l'utilisation de ce symbole permet d'écrire des grammaires dans lesquelles le contexte d'application de la règle (partie *si*) et les contraintes morpho-syntaxiques qu'elle impose (partie *alors*) ne portent pas nécessairement sur les mêmes mots d'une séquence, comme c'est par exemple le cas dans la figure 2.14. Cette grammaire s'interprète de la manière suivante : si on trouve un tiret suivi par *il*, *elle* ou *on* pronom préverbal, alors ce tiret doit être précédé par un verbe, éventuellement suivi de *-t*.

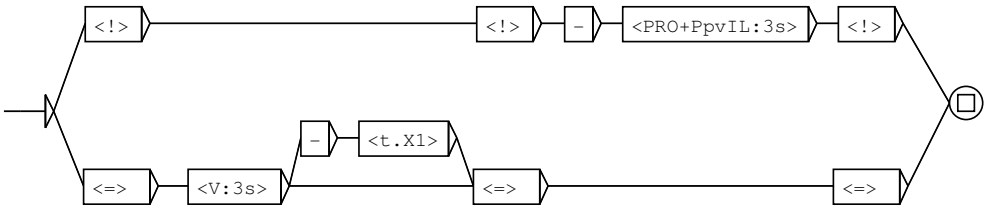

FIG. 2.14 – Utilisation du point de synchronisation

2.7.2 Application

Avant d'être utilisée pour la levée d'ambiguïtés, une grammaire ELAG est d'abord traduite en un automate lexical déterministe qui reconnaît l'ensemble des séquences d'unités lexicales (ou analyses lexicales d'un texte) qui ne sont pas rejetées par la grammaire. Le calcul de cet automate met en œuvre la plupart des opérations sur les automates lexicaux que nous avons définies. Ce calcul est expliqué plus en détail dans [Laporte et Monceaux, 1999]. L'application d'une grammaire à un texte consiste ensuite simplement à calculer l'intersection de l'automate ainsi compilé avec l'automate du texte. Nous obtenons comme résultat un nouvel automate du texte dans lequel toutes les séquences de mots étiquetés rejetées par la grammaire ont été supprimées.

Ce mode de fonctionnement par intersection d'automates a des conséquences intéressantes sur les propriétés des grammaires de désambiguïsation. En particulier, les effets produits par plusieurs grammaires sont cumulatifs et indépendants de leur ordre d'application sur le texte (puisque l'intersection d'automates est une opération commutative). Cette propriété est très utile, car elle permet de cumuler les grammaires écrites par plusieurs linguistes travaillant indépendamment sur des problèmes d'ambiguïtés différents.

2.7.3 Quelques grammaires

Nous présentons ici quelques grammaires ELAG que nous avons écrites pour la désambiguïsation des textes français. La plupart de ces grammaires ont été écrites en collaboration avec Anne Dister et ont fait l'objet d'une communication au 22^e colloque international «Grammaires et Lexiques Comparés».

Les quatre premières grammaires (PpvIL, PpvLE, PpvLUI, PpvPR) s'intéressent aux contextes d'apparition des pronoms pré et post-verbaux du français et spécifient grosso modo les mêmes contraintes :

– soit le pronom clitique est placé à gauche du verbe et, dans ce cas, seuls des pronoms clitiques d'un cas différent peuvent être insérés entre ces deux éléments :

  *Luc le (lui+*la) donne*

Si le pronom considéré est un pronom sujet, alors nous pouvons contraindre l'accord en genre et en nombre entre celui-ci et le verbe conjugué (graphe PpvIL).

– soit le pronom est postposé au verbe et, dans ce cas, on a affaire à une forme à l'impératif et le pronom est relié au verbe par une chaîne de pronoms clitiques séparés par des tirets[1] :

passe-leur la balle !
casse-le-lui !

Si le pronom postposé est un pronom sujet, alors nous avons affaire à une construction interrogative :

Quand viendras-tu ?

Pour ce dernier cas, nous décrivons les contraintes d'accord entre le verbe et son sujet dans une autre grammaire PpvPostPos (figure 2.21). Le graphe PpvSeq (figure 2.22) complète toutes ces grammaires et impose qu'il n'y ait pas plus de deux pronoms clitiques accusatif ou datif qui se suivent[2]. Cette grammaire permet de lever des ambiguïtés dans des phrases telles que

je n'ai pas le temps mais lui te le fera.

où la première forme *lui* doit être étiquetée en tant que pronom tonique et non clitique.

Les grammaires qui suivent traitent de problèmes d'ambiguïté divers mais récurrents dans les textes français.

Ainsi, la grammaire leDET (figure 2.23) traite des déterminants définis, indéfinis, démonstratifs et possessifs qui apparaissent exclusivement en début de syntagme nominal et qui ont donc un contexte gauche très restreint que nous décrivons dans la grammaire.

La grammaire N-PRO (figure 2.24) vient compléter la grammaire PostPos (figure 2.21) pour le cas où la forme du verbe conjugué est ambiguë avec un

[1]Les tirets sont laissés optionnels dans nos grammaires car cette convention n'est pas toujours respectée

[2]Cette grammaire ne contient pas de partie *alors*, elle impose donc que les séquences concordantes avec la partie *si* doivent être rejetées sans condition

adjectif ou un nom. Dans le cas où une telle forme apparaît juste à gauche d'un pronom séparé par un trait d'union, nous supprimons les interprétations nom ou adjectif, comme par exemple dans les phrases :

reste-t-il encore du jambon ?
donne-le moi

La grammaire siADV (figure 2.25) permet de lever certaines ambiguïtés sur la forme *si* qui peut être étiquetée comme adverbe ou conjonction de subordination. Dans le cas où il s'agit d'un adverbe, alors nécessairement cette forme est suivie soit par un adjectif soit par un autre adverbe qu'elle intensifie :

Il parlait si lentement que je me suis assoupi.
Elle était si belle dans sa robe d'été.

La grammaire neVpas (figure 2.26) permet de lever certaines ambiguïtés sur la forme *pas* substantif ou adverbe de négation (composé avec *ne*). Lorsque cette forme apparaît précédée d'un verbe, lui-même précédé par la forme *ne*, alors nous imposons que *pas* ne soit pas étiqueté comme *nom* de manière à ce que seule l'étiquette ADV subsiste.

Enfin, notre dernière grammaire VV (figure 2.27) montre qu'une grammaire ELAG permet d'imposer des contraintes sur des séquences d'une taille non bornée (contrairement au étiqueteurs lexicaux usuels ([Schmid, 1994]) qui travaillent généralement sur des fenêtres de deux ou trois mots). Cette grammaire impose que lorsque l'on a deux verbes conjugués dans la même phrase, alors nécessairement soit les deux verbes sont séparés par une conjonction de coordination ou un symbole de ponctuation, soit une phrase est subordonnée à l'autre, et dans ce cas elle est introduite par une conjonction de subordination ou bien un pronom relatif. Cette grammaire suppose que l'automate auquel elle est appliquée est limité à une phrase.

Nous présentons dans les figures 2.15 et 2.16 le résultat de l'application de toutes ces grammaires sur l'automate de la phrase :

La porte du car se ferme automatiquement

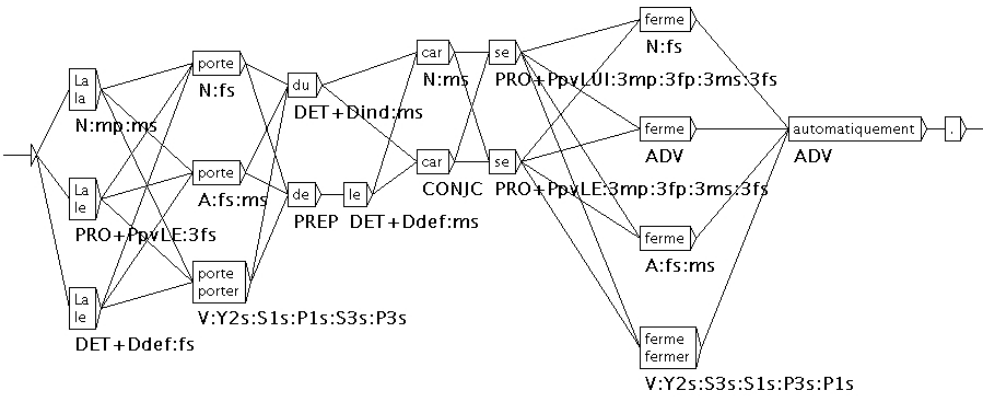

FIG. 2.15 – Automate du texte obtenu après étiquetage morpho-syntaxique

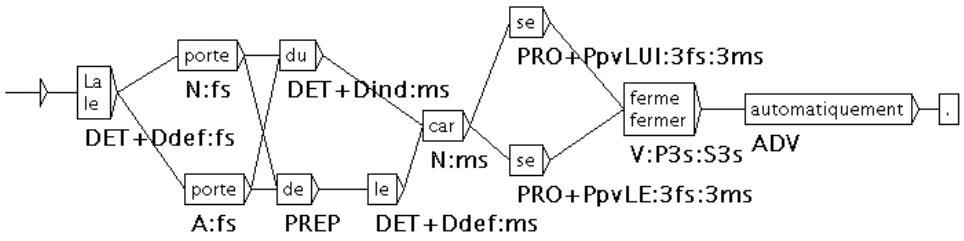

FIG. 2.16 – Automate du texte après application des grammaires ELAG

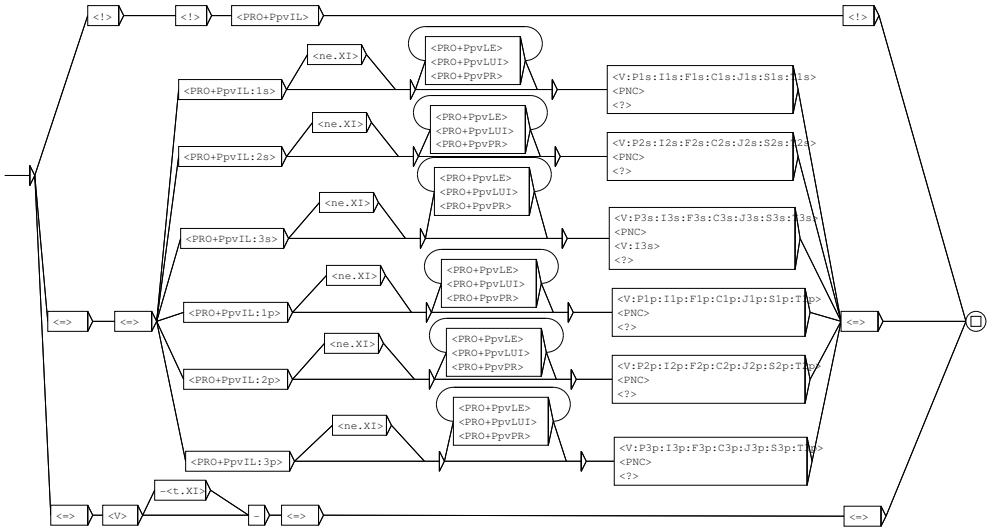

FIG. 2.17 – Grammaire PpvIL

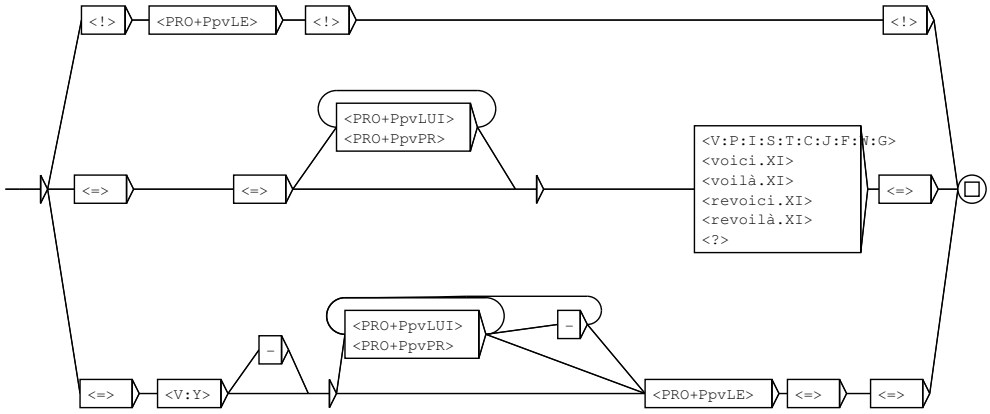

FIG. 2.18 – Grammaire PpvLE

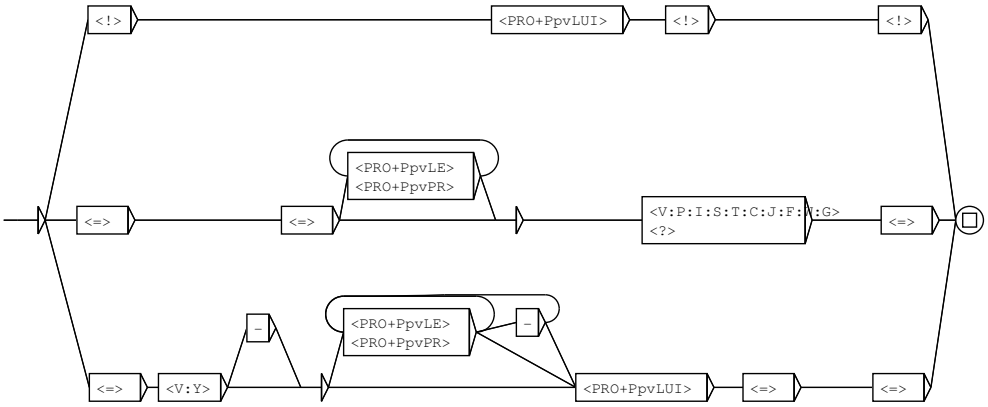

FIG. 2.19 – Grammaire PpvLUI

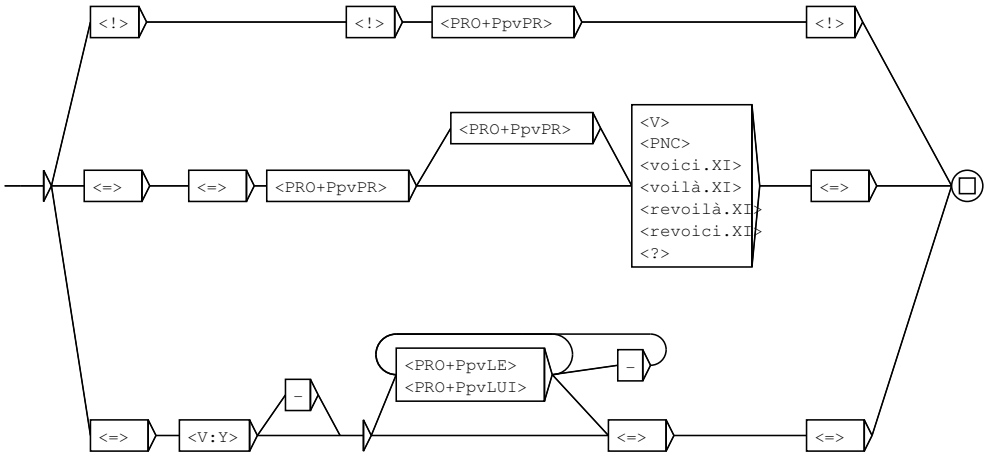

FIG. 2.20 – Grammaire PpvPR

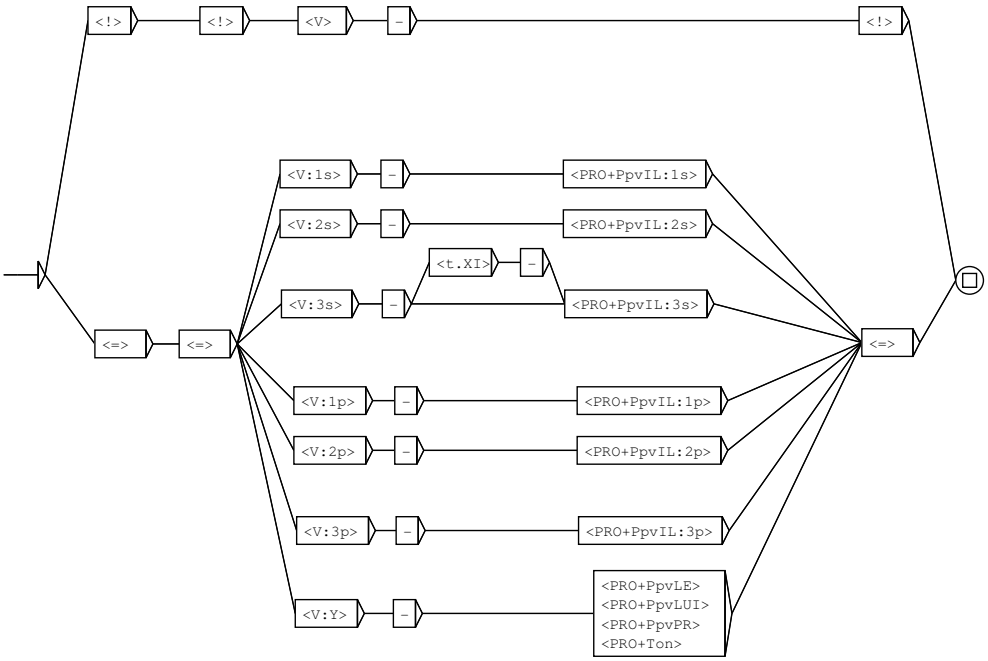

FIG. 2.21 – Grammaire PpvPostpos

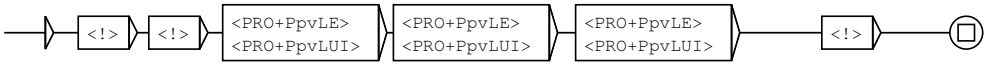

FIG. 2.22 – Grammaire PpvSeq

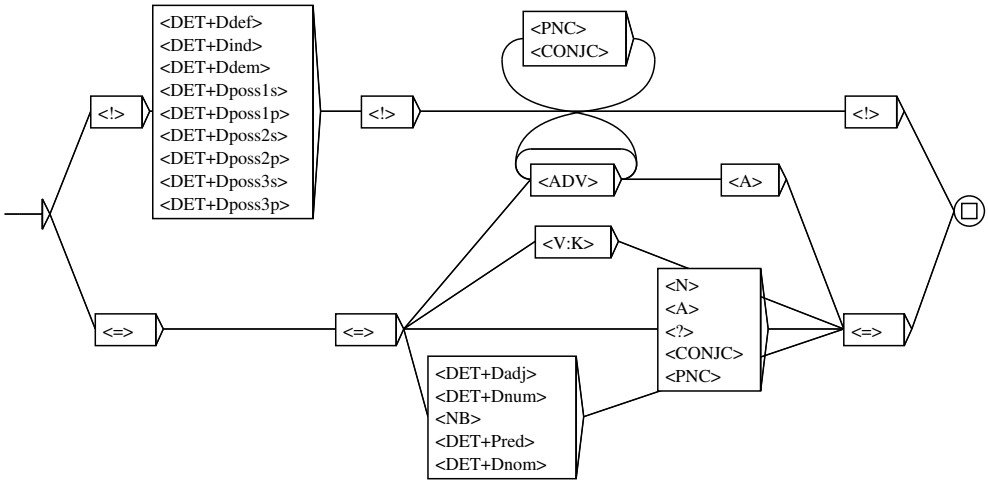

FIG. 2.23 – Grammaire leDET

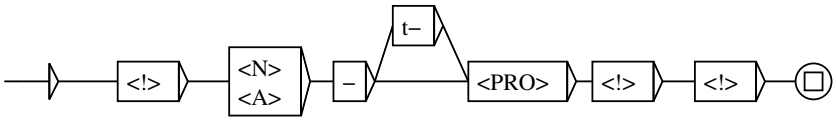

FIG. 2.24 – Grammaire N-PRO

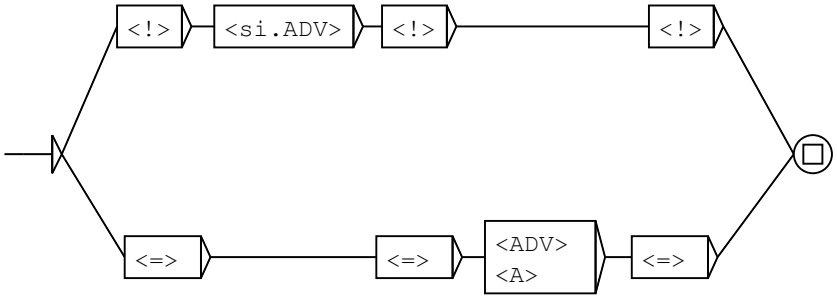

FIG. 2.25 – Grammaire siADV

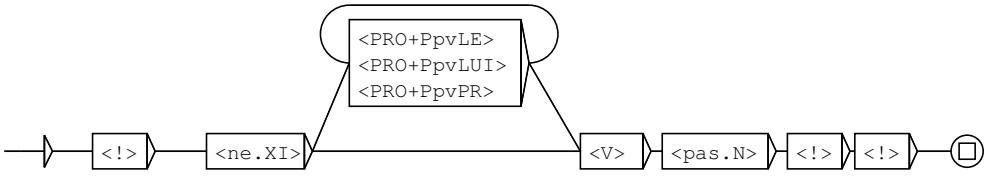

FIG. 2.26 – Grammaire neVpas

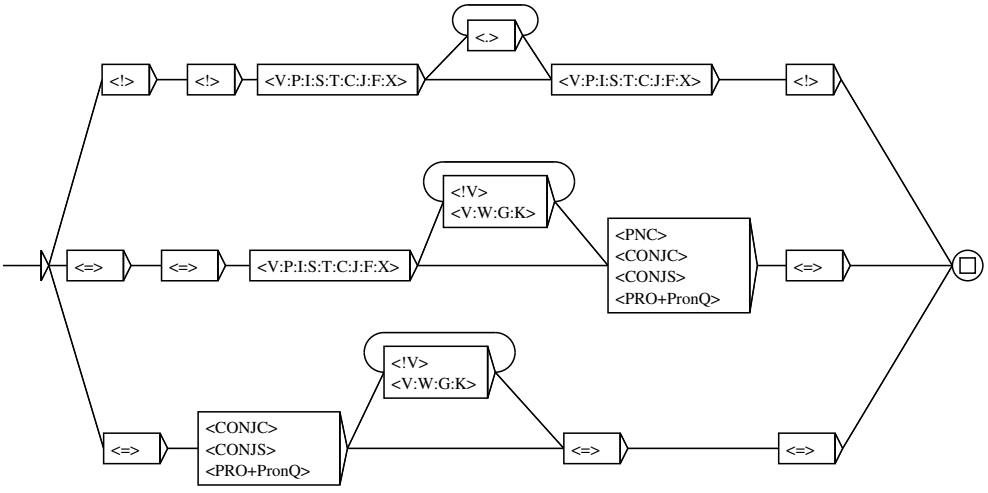

FIG. 2.27 – Grammaire VV

2.8 Le cas des transducteurs

Nous avons vu que le modèle des automates lexicaux est adéquat pour la représentation des textes étiquetés morpho-syntaxiquement ainsi que pour la représentation des grammaires de désambiguïsation. Les automates lexicaux sont également adaptés pour rechercher efficacement dans les textes des séquences concordant avec les formes linguistiques décrites dans des grammaires locales. Cependant, ce modèle n'est pas à proprement parler adapté à l'analyse de textes, dans le sens où un automate fini ne fait que de la reconnaissance de formes et ne permet pas d'associer une analyse aux segments reconnus par la grammaire. C'est pourquoi une grammaire locale peut être également un transducteur, c'est-à-dire comporter des sorties sous la forme de chaînes de caractères. Lors de l'application de ce type de grammaire sur un texte, ces sorties sont insérées dans le texte en entrée et permettent ainsi d'annoter les segments identifiés ou de formaliser des relations entre plusieurs segments décrits par la grammaire. Par exemple, le graphe de la figure 2.28 présente la grammaire locale de la figure 2.1 (page 15) à laquelle nous avons rajouté des sorties (qui apparaissent sous les boîtes) permettant de baliser dans les textes les séquences verbales à l'indicatif reconnues par la grammaire.

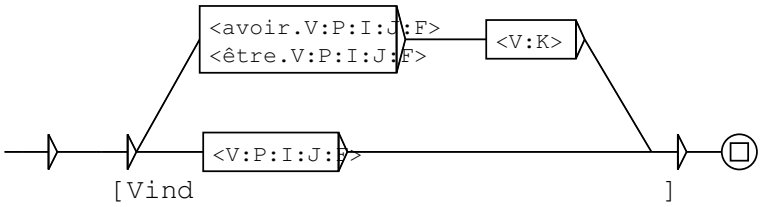

FIG. 2.28 – Grammaire locale avec sorties

L'application de cette grammaire, de façon glissante, sur le texte :

Tandis que Lea étirait son corps au soleil, Luc est parti travailler.

produira ainsi la sortie suivante, dans laquelle les verbes conjugués au mode indicatif ont été identifiés et balisés :

Tandis que Lea [Vind prélassait] son corps au soleil, Luc [Vind est parti] travailler.

Ce type de transducteurs finis est utilisé pour des situations variées qui ne nécessitent généralement qu'une analyse superficielle du texte traité dans le sens où seuls les segments pertinents pour l'application sont analysés. Les grammaires locales avec sorties ont ainsi été à la base d'applications dans des traitements divers tels que l'extraction d'informations [Poibeau, 2001, Nakamura, 2005], la reconnaissance d'entités nommées [Friburger, 2002, Krstev *et al.*, 2005], ou encore l'identification de structures grammaticales [Mason, 2004, Danlos, 2005a]. Les transducteurs finis ont également été utilisés pour faire de l'analyse syntaxique profonde de texte dans des traitements mettant en oeuvre des applications en cascade de transducteurs [Roche, 1993, Abney, 1996, Joshi et Hopely, 1996].

Dans cette section, nous décrivons comment nous avons étendu notre algorithme de déterminisation des automates lexicaux au cas des automates lexicaux avec sortie. Nous présentons d'abord l'algorithme de déterminisation d'un transducteur fini classique et nous montrons que nous ne pouvons pas l'utiliser directement pour la déterminisation d'une grammaire locale avec sorties puisque son application peut modifier le comportement de la grammaire, notamment vis-à-vis de la position dans le texte où sont insérées les sorties. Ceci nous a amené à développer une version modifiée de cet algorithme, que nous présentons ensuite. De plus, le fait que tous les transducteurs ne sont pas déterminisables nous a conduit à implémenter une version paresseuse de notre algorithme qui accepte en entrée tous les transducteurs lexicaux. Nous terminons ce chapitre en donnant des évaluations des performances de nos programmes en présentant le cas de l'application sur une année du corpus Le Monde des grammaires de L. Danlos pour la reconnaissance et l'étiquetage du pronom *il* dans son emploi impersonnel ou anaphorique [Danlos, 2005a, Danlos, 2005b].

2.8.1 Algorithme de déterminisation classique

Dans [Choffrut, 1977] et [Choffrut, 1978], C. Choffrut s'intéresse à caractériser la classes des transducteurs finis réalisant des fonctions sous-séquentielles

et qui sont par conséquent déterminisables. La démonstration contient un algorithme implicite qui permet effectivement de construire un transducteur déterministe à partir d'un transducteur réalisant une fonction sous-séquentielle. Cet algorithme a ensuite été décrit de manière plus explicite dans différents ouvrages [Berstel, 1979] [Mohri, 1997] [Roche et Schabes, 1997], dans lesquels on trouve des extensions à différentes classes de transducteurs, notamment les transducteurs p-séquentiels, dont l'entrée est déterministe mais qui admettent plusieurs sorties pour une même séquence en entrée. Ce modèle est souvent utilisé en TAL, puisqu'il permet de prendre en compte les différents résultats produits par l'analyse de séquences ambiguës. Nous donnons ici les grandes lignes de cet algorithme, avant d'aborder le cas plus complexe des transducteurs lexicaux.

Formellement, nous définissons un transducteur fini T comme un tuple $T = (Q, \Sigma, \Delta, I, F, E)$ où :

- Q est un ensemble fini d'état ;
- Σ est l'alphabet d'entrée constitué d'un ensemble fini de symboles ;
- Δ est l'alphabet de sortie constitué d'un ensemble fini de symboles ;
- $I \subseteq Q$ est l'ensemble des états initiaux ;
- $F \subseteq Q$ est l'ensemble des états finaux ;
- $E \subset Q \times \Sigma \times \Delta^\star \times Q$ est un ensemble des transitions.

Nous définissons formellement un transducteur fini déterministe comme un tuple $T = (Q, \Sigma, \Delta, i, F, \delta, \sigma, \rho)$ où :

- Q est un ensemble fini d'états ;
- Σ est l'alphabet d'entrée ;
- Δ est l'alphabet de sortie ;
- $i \in Q$ est l'état initial ;
- $F \subseteq Q$ est l'ensemble des états finaux ;
- $\delta : Q \times \Sigma \mapsto Q$ est la fonction de transition qui associe à un état de départ et un symbole en entrée un état d'arrivée ;
- $\sigma : Q \times \Sigma \mapsto \Delta^\star$ est la fonction de sortie qui associe à un état de départ et un symbole en entrée une production en sortie ;
- $\rho : F \mapsto 2^{\Delta^\star}$ la fonction qui associe à chaque état final, un ensemble de sorties à émettre lorsque l'analyse se termine dans cet état. Si pour tout $q \in F$ le cardinal de l'ensemble $\rho(q)$ est borné par une constante p, alors

T est un transducteur p-séquentiel.

La déterminisation d'un transducteur fini est très proche de la construction de l'automate des sous ensembles utilisé pour la déterminisation des automates sans sortie (cf. section 2.6.3). La principale différence est que, pour le cas des transducteurs, on associe à chaque état d'un sous-ensemble les sorties qui ont été retardées du fait que différentes sorties peuvent être émises à la lecture d'un même symbole en entrée par un transducteur fini non déterministe.

Étant donné un transducteur $T = (Q, \Sigma, \Delta, I, F, E)$ qui réalise une relation p-séquentielle, nous construisons un transducteur p-séquentiel $D = (Q', \Sigma, \Delta, i, F', \delta, \sigma, \rho)$ équivalent de la manière suivante.

Chaque état de D est un ensemble de couples $(q, \omega) \in Q \times \Delta^*$. L'état initial i de D est défini par

$$i = \{(q, \epsilon) \mid q \in I\}$$

On calcule, les fonctions de transition δ et de sortie σ comme suit. Etant donné un état P de D, pour chaque symbole a de Σ, on définit l'ensemble

$$R(P, a) = \{(q', wu) \in Q \times \Delta^* \mid (q, w) \in P \text{ et } (q, a, u, q') \in E\}$$

Si cet ensemble est vide, D n'a pas de transition sortante de P étiquetée en entrée par a.

Dans le cas contraire, on calcule v, le plus long préfixe commun des mots vw pour tout $(q, vw) \in R(P, a)$. C'est ce mot v qui va étiqueter en sortie la transition sortante de P étiquetée par a en entrée. On ajoute alors à D un nouvel état P', composé des états d'arrivée q' auxquels sont associées les sorties retardées, c'est-à-dire les suffixes w qui ne sont pas émis dans la transition sortante :

$$P' = \{(q', w) \mid (q', vw) \in R(P, a)\}$$

On définit la transition de P vers P' étiquetée par a en entrée et par v en sortie.

$$\delta(P, a) = P' \qquad \sigma(P, a) = v$$

Enfin, on définit $\rho(P)$ qui associe à l'état P un ensemble de sorties finales comme suit :

$$\rho(P) = \{w \mid (q, w) \in P \text{ et } q \in F\}$$

On continue ainsi l'algorithme tant qu'il reste de nouveaux états à traiter.

L'ensemble des états finaux de D correspond à l'ensemble des états P tels que $|\rho(P)| \geq 1$.

Si pour tout état final q de D, $|\rho(q)| = 1$, alors le transducteur D ainsi construit est sous-séquentiel, c'est-à-dire qu'il a une entrée déterministe et associe au plus une sortie pour toute entrée.

S'il existe un nombre p tel que, pour tout $q \in Q'$, $|\rho(q)| \leq p$, alors D est dit p-séquentiel, c'est-à-dire qu'il a une entrée déterministe et qu'il associe au plus p sorties pour toute entrée.

Dans le cas contraire, le transducteur T passé en entrée à notre algorithme ne réalise pas une relation p-séquentielle, et l'algorithme de déterminisation ne se termine pas : il génère un nombre infini d'états.

Par exemple, la figure 2.29 présente un transducteur T_1 non déterministe et la figure 2.30 présente le transducteur déterministe T_2 équivalent obtenu par déterminisation de T_1. Les sous-ensembles correspondant à chaque état de T_2 sont laissés apparents dans la figure.

Le transducteur T_1 a deux transitions sortantes de l'état 0 étiquetés par a en entrée mais qui ont des sorties différentes. Ainsi, lors de la construction du transducteur déterministe T_2, la transition sortante de l'état 0 étiquetée par a en entrée contient une sortie vide ; les deux sorties sont retardées dans l'état 1 ayant pour sous-ensemble correspondant $\{(1, b), (2, a)\}$. Ces sorties seront émises dans les transitions sortantes de l'état 1 en fonction du prochain symbole en entrée.

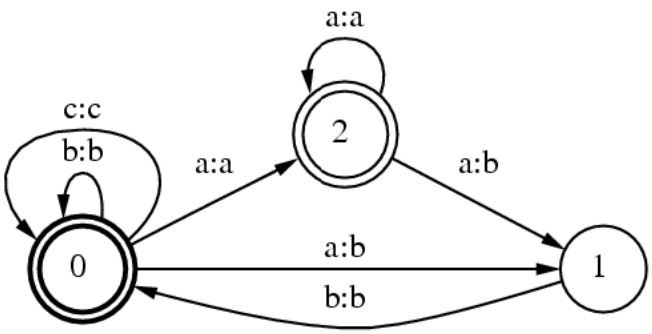

FIG. 2.29 – Transducteur T_1 non déterministe

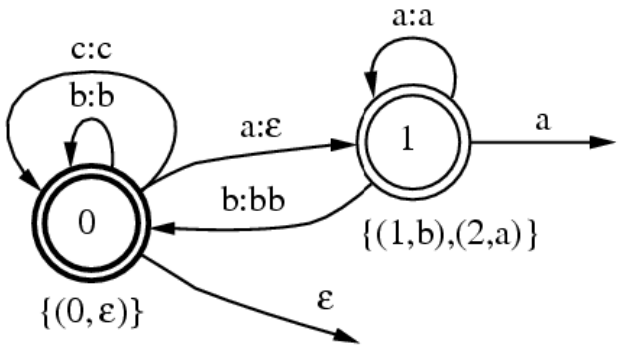

FIG. 2.30 – Transducteur sous-séquentiel T_2 équivalent au transducteur T_1

2.8.2 Application aux grammaires locales

L'application d'une grammaire locale avec sorties diffère substantiellement de l'application d'un transducteur fini classique. En effet, lors de l'application d'une grammaire locale sur un texte, les sorties sont insérées dans le texte en entrée au sein des séquences reconnues, ce qui produit comme résultat le texte original annoté de nouveaux éléments. Avec une transduction classique, le résultat consiste en un nouveau texte composé exclusivement de

la concaténation des sorties de la grammaire. Du fait de cette particularité, nous ne pouvons pas directement utiliser l'algorithme précédent pour déterminiser les grammaires locales. En effet, le fait de retarder les sorties lors de la construction du transducteur déterministe modifie la position où celles-ci seront émises dans le texte si bien que la grammaire déterministe résultat aura un comportement différent de la grammaire originale.

Ainsi, considérons la grammaire locale de la figure 2.31 qui est également présentée sous sa forme duale dans la figure 2.32.

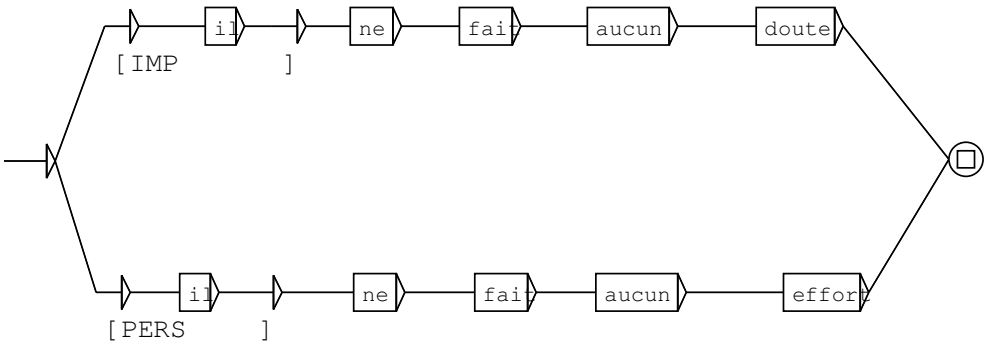

FIG. 2.31 – Grammaire locale avec sorties

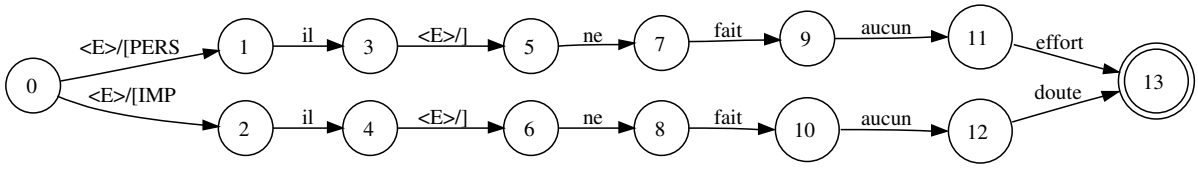

FIG. 2.32 – Transducteur lexical non déterministe

La grammaire reconnaît les deux séquences suivantes :

il ne fait aucun doute
il ne fait aucun effort

et produit en sortie le même texte dans lequel le pronom *il* est balisé par les étiquettes IMP et PERS respectivement :

[IMP il] ne fait aucun doute
[PERS il] ne fait aucun effort

Or, si nous appliquons l'algorithme de déterminisation sur cette grammaire nous obtenons le transducteur lexical de la figure 2.33.

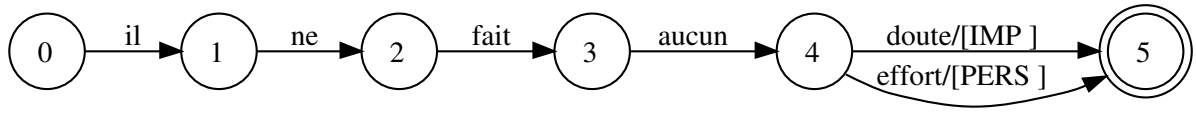

FIG. 2.33 – Résultat de la déterminisation

Le résultat est bien une grammaire locale déterministe, cependant cette dernière n'est clairement pas équivalente à la grammaire originale, puisque son application sur les deux mêmes séquences que précédemment donnera les résultat suivants[3] :

il ne fait aucun [IMP] doute
il ne fait aucun [PERS] effort

2.8.3 Déterminisation des grammaires locales avec sorties

Pour pallier ce problème de désynchronisation des entrées avec les sorties qui apparaît lors de la déterminisation des grammaires locales, nous avons été amené à modifier le modèle des transducteurs finis représentant une grammaire locale déterministe. Au lieu de simples chaînes de caractères, nous représentons chaque sortie par un ensemble de couples formés d'une chaîne de caractères et d'un entier ; ce dernier décrit le décalage de la sortie par rapport à sa position dans la grammaire originale. Tous ces décalages sont

[3]Lorsqu'une sortie est présente dans une transition qui n'est pas étiquetée par le mot vide, la convention est d'émettre la chaîne en sortie à gauche du token dans le texte qui a permis le franchissement de cette transition.

initialement à zéro. A chaque fois qu'une sortie est retardée lors de la déterminisation de la grammaire, on incrémente son décalage de manière à ne pas perdre la position à laquelle elle doit être émise lors de son application sur un texte.

Nous avons ainsi implémenté l'algorithme de déterminisation d'un transducteur lexical qui construit un transducteur lexical déterministe dans ce nouveau formalisme (algorithme 5).

L'algorithme prend en entrée un transducteur lexical non déterministe $T_1 = (Q_1, Lex, \Delta, I_1, F_1, E_1)$ où

- Q_1 est un ensemble fini d'état ;
- Lex, l'ensemble des masques lexicaux sur le jeu d'étiquettes utilisé ;
- Δ, l'alphabet de sortie ; dans le cadre de nos applications, les sorties sont des chaînes de caractères Unicode ;
- $I_1 \subseteq Q_1$, l'ensemble des états initiaux ;
- $F_1 \subseteq Q_1$, l'ensemble des états finaux ;
- $E_1 \subset Q_1 \times Lex \times \Delta^\star \times Q_1$, l'ensemble de transitions.

Il retourne un transducteur lexical déterministe équivalent $T_2 = (Q_2, Lex, \Delta, i_2, F_2, \delta, \sigma, \rho)$ avec

- $Q_2 \subset 2^{Q_1 \times 2^{\Delta^\star \times N}}$, l'ensemble des états de T_2 ;
- i_2, l'état initial ;
- F_2, l'ensemble des états finaux de T_2 ;
- $\delta : Q_2 \times Lex \mapsto Q_2$, la fonction de transition ;
- $\sigma : Q_2 \times Lex \mapsto 2^{\Delta^\star \times N}$, la fonction de sortie ; les sorties qui étiquettent les transitions de T_2 sont des ensembles de couples $(w, i) \in \Delta^\star \times N$ où w est la chaîne de caractères à émettre et i spécifie son décalage par rapport à sa position dans le transducteur original ;
- $\rho : F_2 \rightarrow 2^{\Delta^\star \times N}$, la fonction des sorties finales qui associe des sorties supplémentaires aux états finaux.

Les principales modifications par rapport à l'algorithme original découlent directement des différences sur les alphabets d'entrée et sortie entre les transducteurs lexicaux et les transducteurs finis.

entrée: $T_1 = (Q_1, Lex, \Delta, I_1, F_1, E_1)$, un transducteur lexical non déterministe
sortie: $T_2 = (Q_2, Lex, \Delta, i_2, F_2, \delta, \sigma, \rho)$, un transducteur lexical déterministe

1: $F_2 \leftarrow \emptyset$
2: $i_2 \leftarrow \bigcup_{i \in I_1} \{(i, \emptyset)\}$
3: $Q \leftarrow \{i_2\}$
4: **tant que** $Q \neq \emptyset$ **faire**
5: $q_2 \leftarrow$ DEQUEUE(Q)
6: **si** il existe $(q, o) \in q_2$ tel que $q \in F_1$ **alors**
7: $F_2 \leftarrow F_2 \cup \{q_2\}$
8: $\rho(q_2) \leftarrow \{o \mid (q, o) \in q_2 \text{ et } q \in F_1\}$
9: **fin si**
10: $Trans =$ DECOUPE_TRANS$(\{(q, m, w \cup \{(v, 0)\}, q') \mid (q, w) \in q_2 \text{ et } (q, m, v, q') \in E_1\})$
11: **pour tout** m tel que $(q, m, o, q') \in Trans$ **faire**
12: $v \leftarrow \bigcap_{(q,m,w,q') \in Trans}(w)$
13: $\sigma(q_2, m) \leftarrow v$
14: $\delta(q_2, m) \leftarrow \bigcup_{(q,m,v \cup w,q') \in Trans}\{(q', delay(w))\}$
15: **si** $\delta(q_2, m)$ est un nouvel état **alors**
16: ENQUEUE$(Q, \delta(q_2, m))$
17: **fin si**
18: **fin pour**
19: **fin tant que**

Algorithme 5: Déterminisation d'un transducteur lexical

Ainsi, à la ligne 10, de la même manière que pour la déterminisation des automates lexicaux sans sortie, nous découpons les transitions sortantes des états considérés de manière à ce que les masques lexicaux qui les étiquettent soient deux à deux disjoints ou égaux. La sortie de chacune de ces transitions est formée de l'ensemble des sorties retardées qui sont associées à l'état de départ de la transition auquel nous rajoutons la sortie de la transition dans la grammaire originale avec un décalage de 0 (ligne 10).

Dans les lignes 11 à 20, nous itérons sur tous les masques lexicaux m qui étiquettent au moins une des transitions découpées. Pour chaque masque m,

nous calculons (ligne 12) $v \subset \Sigma^\star \times N$, l'ensemble des sorties retardées qui sont présentes dans toutes les sorties des transitions étiquetées en entrée par m (c'est-à-dire l'intersection de tous ces ensembles). Cet ensemble v constitue la sortie de la transition sortante de q_2 étiquetée par m en entrée dans le transducteur déterministe (ligne 13). Ce calcul de v est analogue au calcul du plus long préfixe commun que l'on retrouve lors de la déterminisation des transducteurs finis (cf. section 2.8.1), seulement ici, chaque sortie est associée à un décalage qui spécifie la position à laquelle elle doit être émise ; nous pouvons, de ce fait, étiqueter en sortie la transition déterministe par une sortie décalée même si des sorties qui seront émises plus tôt dans le résultat (i.e. avec un décalage supérieur) sont retardées.

Ligne 14, nous calculons l'état de destination de la transition dans le transducteur déterministe. Celui-ci consiste en l'ensemble des états d'arrivée des transitions étiquetées par m, auxquels on associe les sorties décalées privées des sorties déjà émises dans la transition déterministe ; le décalage de ces sorties restantes est alors incrémenté de 1 (appel à la fonction *delay*).

Enfin, nous rajoutons l'état de destination dans la pile Q s'il apparaît pour la première fois (ligne 15 et 16), de manière à ce qu'il soit traité lors des prochaines itérations. L'algorithme se termine lorsque cette pile est vide. Dans le cas où le transducteur lexical T_1 en entrée n'a pas de transducteur p-séquentiel équivalent, l'algorithme ne se termine pas et génère un nombre infini d'états.

La figure 2.34 présente par exemple le transducteur lexical de la figure 2.32 correctement déterminisé.

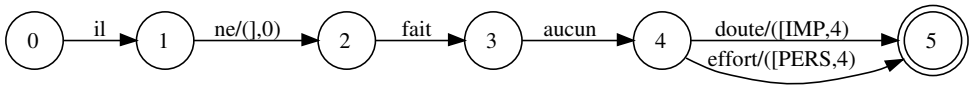

FIG. 2.34 – Transducteur lexical déterministe

Notons qu'il est nécessaire d'utiliser une pile lors de l'application d'un transducteur lexical déterministe sur un texte afin de stocker les symboles d'entrée

et sortie lorsque ils apparaissent durant l'analyse, de manière à pouvoir les émettre dans le bon ordre une fois la reconnaissance terminée. Cependant la grammaire obtenue est bien déterministe et il n'est donc pas nécessaire de mettre en place des procédures coûteuses de *backtracking* (retour en arrière) pour explorer toutes les possibilités lors de son application sur un texte, comme on aurait été obligé de le faire en appliquant la grammaire non déterministe de la figure 2.32.

Notons enfin que dans [van Noord et Gerdemann, 2001], Van-Noord présente le modèle des transducteurs finis sur prédicats, une extension du modèle des transducteurs finis adapté au traitement des langues très proche de notre modèle des transducteurs lexicaux. Son modèle est plus général que le nôtre puisque les symboles qui étiquettent les transitions en entrée sont présentés comme de simples prédicats décrivant un sous-ensemble de l'alphabet d'entrée et leurs formes ne sont pas plus spécifiées. En ce sens, nous pouvons considérer notre modèle comme une instance particulière du modèle des transducteurs sur prédicats dans lequel les étiquettes (les masques lexicaux) ont une forme mieux spécifiée. Dans son article, Van-Noord propose également un algorithme, un peu différent du nôtre, pour déterminiser ce type de transducteurs qui prend également en compte la possibilité de réémettre les symbole du texte en entrée en combinaison avec les sorties de la grammaire. Comme avec nos transducteurs lexicaux déterministes, les transducteurs sur les prédicats déterminisés nécessitent également d'utiliser une pile durant l'analyse afin d'émettre les symboles en sortie dans le bon ordre par rapport à l'ordre dans lequel ils sont récupérés.

2.8.4 Déterminisation paresseuse

Comme nous l'avons évoqué précédemment, tous les transducteurs finis ne sont pas déterminisables ; il existe des transducteurs pour lesquels l'algorithme de déterminisation ne s'arrête pas. Par exemple, le transducteur simple de la figure 2.35 ne peut pas être déterminisé.

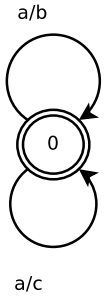

FIG. 2.35 – Transducteur fini non déterminisable

En effet, à chaque étape, ce transducteur accepte un a en entrée et peut émettre indifféremment soit le symbole b soit le symbole c en sortie. Ainsi, lors de la construction du transducteur déterministe équivalent, les sorties sont repoussées indéfiniment au niveau des états finaux ce qui créé un transducteur à nombre infini d'états (figure 2.36).

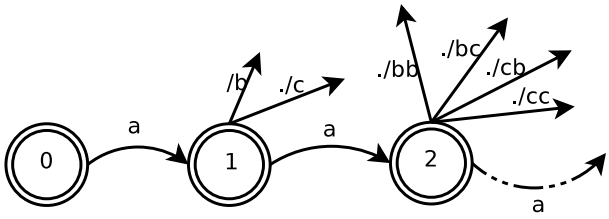

FIG. 2.36 – Transducteur déterministe à nombre infini d'états

Il en est de même pour les grammaires locales avec sorties. Toutes ne sont pas déterminisables. Par exemple, les séquences *V Prep N Prep N . . . Prep N* sont extrêmement ambiguës en français : chaque *Prep N* peut être attaché soit au verbe (en tant qu'argument ou adverbe) soit à un des noms à sa gauche. Le nombre d'interprétations possibles est de l'ordre de la factorielle sur la longueur de la séquence. Ainsi, considérons par exemple la grammaire de la figure 2.37, qui identifie les arguments du verbe *parler* dans certaines constructions simples.

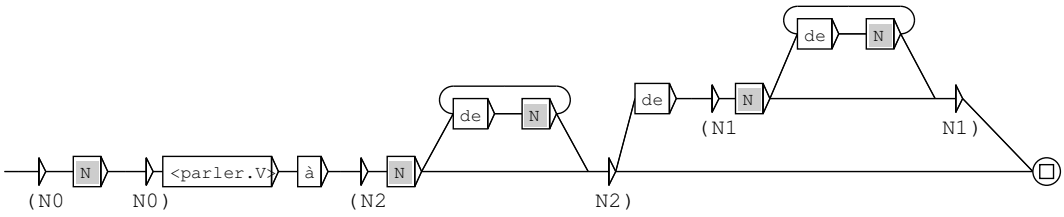

FIG. 2.37 – Grammaire locale avec sorties non déterminisable

Cette grammaire produit un nombre d'analyses potentiellement non borné en fonction de la longueur de la séquence en entrée :

(N0 Luc N0) parle à (N2 la voisine de son fils N2)
(N0 Luc N0) parle à (N2 la voisine N2) de (N1 son fils N1)

(N0 Luc N0) parle à (N2 la voisine de l'avocat de Lea N2)
(N0 Luc N0) parle à (N2 la voisine de l'avocat N2) de (N1 Lea N1)
(N0 Luc N0) parle à (N2 la voisine N2) de (N1 l'avocat de Lea N1)
etc.

La grammaire n'est donc pas déterminisable et nous ne pouvons donc pas lui appliquer notre algorithme de déterminisation tel que nous l'avons défini.

Afin de pouvoir optimiser l'application de tous types de grammaire, nous avons implémenté une version paresseuse de notre algorithme de déterminisation d'un transducteur lexical que nous appliquons au moment de la confrontation de la grammaire avec un texte. Nous déterminisons localement la grammaire à la demande lors de l'exploration des chemins de la grammaire qui concordent avec des séquences dans l'automate du texte. Ainsi, nous explorons (et déterminisons) uniquement les chemins dans la grammaire qui nous intéressent et nous n'explorons pas ceux qui ne correspondent avec aucune séquence du texte traité. Nous savons que cette exploration est limitée (et donc que notre algorithme se termine) puisque le texte traité en entrée est de taille finie. De plus, nous traitons des textes préalablement découpés en phrases et nous appliquons notre transducteur lexical sur le texte une phrase après l'autre. Ainsi, la profondeur maximale d'exploration lors de la construction de la grammaire déterministe ne peut pas, dans le pire des cas, dépasser

la longueur de la phrase (c'est-à-dire une vingtaine de mots en moyenne).

A la fin de l'application d'une grammaire sur un texte, nous sauvegardons le transducteur partiellement déterminisé de manière à pouvoir le réutiliser pour l'analyse de nouveaux textes sans avoir à recalculer sa partie déjà déterminisée. De ce fait, plus une grammaire est utilisée pour l'analyse de textes différents et plus la grammaire est optimisée. Les performances sont donc susceptibles d'être améliorées au fur et à mesure des traitements successifs.

2.8.5 Évaluations des performances

Nous présentons ici des évaluations des performances de notre programme qui applique un transducteur lexical sur un automate du texte, produisant en sortie un texte annoté. En guise de comparaison, nous donnons les temps obtenus en procédant aux mêmes traitements avec le programme équivalent d'Unitex.

Nous avons procédé à quatre tests, que nous pensons représentatifs. Les deux premiers ont été effectués en utilisant la grammaire de Laurence Danlos pour la reconnaissance et l'étiquetage du pronom *il* dans les constructions impersonnelles [Danlos, 2005a]. La grammaire est constituée de 71 graphes différents, qui ne contiennent pas d'appels à des sous-graphes récursifs. Nous avons applati cette grammaire en un unique transducteur lexical non déterministe composé de 8263 états et 26862 transitions. Nous l'avons ensuite optimisée en lui appliquant les opérations de suppression des transition étiquetées par le mot vide, d'émondation et de minimisation. Le transducteur lexical obtenu contient 2299 états et 25432 transitions.

Pour notre premier test, nous avons appliqué cette grammaire sur un extrait de l'année 1994 du journal Le Monde, composé de l'ensemble des phrases qui contiennent au moins une occurrence de la forme *il*. Le corpus obtenu fait une taille de 18 Mo (encodé en UTF-8). Il a été ségmenté lors des opérations de pré-traitement en 107356 phrases et 3 683 266 tokens. Les temps de traitements pour ce test avec le logiciel Unitex nous ayant semblé démesurément

longs[4], nous avons procédé aux tests suivants afin d'obtenir une meilleure comparaison.

Pour le second test, nous avons appliqué la même grammaire sur le roman *Le tour du monde en 80 jours* de Jules Verne (428 Ko), qui a été segmenté en 4 650 phrases et 91 402 tokens.

Pour le troisième test, nous avons appliqué sur notre premier corpus, un sous-ensemble simple de la grammaire de Laurence Danlos, qui reconnaît les constructions de la forme : *il être adj* telles que

il est (important+nécessaire+vraisemblable) que

Enfin pour notre dernier test, nous avons appliqué, toujours sur le même corpus, une grammaire fortement lexicalisée composée de 141 graphes décrivant les déterminants simples et composés du français qui a été récement développée par Anastasia Yannacopoulou et Eric Laporte, à partir de grammaires originales écrites par Maurice Gross [Gross, 2001] et Max Silberztein [Silberztein, 2003] [Silberztein, 2004].

Tous les temps de traitement sont reportés dans le tableau suivant.

	ilIMP	ilêtreAdj	DET	ilIMP
États	2 299	144	3 824	2 299
Transitions	26 862	5 006	94 703	26862
Taille du corpus	18 Mo	18 Mo	18 Mo	428 Ko
Temps avec notre programme (première itération)	247,23s	91s	3min56s	34,6s
Temps avec notre programme (seconde itération)	111,84s	90s	3min29s	2,51s
Temps avec Unitex	plus de 2 semaines	4h28min	33min5s	2,3s

Nous observons que les bénéfices apportés par la déterminisation de la gram-

[4]Nous avons interrompu le programme d'analyse après deux semaines d'exécution alors qu'il indiquait avoir traité 86% du texte.

maire pendant l'analyse dépendent fortement de la structure de celle-ci. Par exemple, la grammaire ilIMP comprend beaucoup d'étiquettes incomplètes qui s'intersectent les unes avec les autres, ainsi les bénéfices de la déterminisation sont nettes. La grammaire des déterminants, quant à elle, est fortement lexicalisée et la déterminisation simple (sans procéder à un découpage des étiquettes) suffit à déterminiser la majeure partie de cette grammaire ; ainsi la déterminisation à la volée de cette grammaire en procédant à un découpage des étiquettes n'apporte pas d'amélioration nette sur les performances de l'analyse.

Enfin, nos programmes semblent mieux adaptés pour le traitement de corpus de grandes tailles, tandis que le programme d'Unitex est plus performant sur de plus petits corpus ou lorsque la grammaire reconnaît des portions très faibles du texte en entrée.

Chapitre 3

Outilex et grammaires WRTN

3.1 Introduction

Ce travail de thèse a été financé par le ministère de l'Industrie dans le cadre du projet RNTL/Outilex. Le projet Outilex vise à mettre à la disposition de la recherche, du développement et de l'industrie une plate-forme logicielle ouverte dédiée au traitement automatique des langues naturelles incluant des outils, des dictionnaires électroniques et des grammaires. La plupart des traitement offerts par la plate-forme sont basés sur l'utilisation d'automates finis et sont adaptés pour l'utilisation de ressources linguistiques à large couverture. Les ressources linguistiques traitées (corpus, dictionnaires et grammaires) sont formatées dans des formats XML qui suivent les recommandations actuelles de définition de normes en matière de modèles de ressources linguistiques. La plate-forme sera ouverte au public en mars 2007, sous une licence peu restrictive (LGPL[1] pour les composants logiciels et LGPL-LR [2] pour les ressources linguistiques) adaptée pour le développement d'applications académiques et industrielles.

[1]Lesser General Public License, `http://www.gnu.org/copyleft/lesser.html`.

[2]Lesser General Public License for Language Resources, http ://infolingu.univ-mlv.fr/lgpllr.html. Les droits et devoirs donnés aux utilisateurs par la licence LGPL-LR sont l'équivalent, pour les ressources linguistiques, de ceux donnés aux utilisateurs de la licence LGPL pour les logiciels.

Nous présentons dans un premier temps les principaux résultats d'Outilex : nous décrivons l'architecture générale de la plate-forme et nous présentons les différents modules proposés par celle-ci, qui implémentent les opérations fondamentales pour le traitement de textes écrits : segmentation du texte, étiquetage morpho-syntaxique, traitements par grammaires et gestion des ressources linguistiques.

La seconde partie de ce chapitre est consacrée à une présentation plus en détail des modules consacrés aux différents traitements textuels qui mettent en oeuvre l'application d'une grammaire sur un texte (module d'analyse et modules qui produisent des résultats sous différentes formes). Nous présentons pour ce faire le formalisme des grammaires WRTN utilisées dans Outilex ; il s'agit d'une extension du formalisme des RTN [Woods, 1970] auquel nous avons ajouté la possibilité de pondérer les transitions des automates. Ce système de pondération permet de favoriser certaines analyses dans le cas où une grammaire ambiguë donnerait plusieurs analyses candidates pour une même entrée. D'autre part, ce formalisme grammatical permet également de faciliter la mise en place de traitements hybrides mélangeant l'utilisation de ressources linguistiques avec des méthodes statistiques.

Nous décrivons ensuite notre algorithme d'analyse permettant d'appliquer une grammaire WRTN sur l'automate du texte ; notre algorithme est inspiré de l'algorithme d'Earley que nous avons adapté de manière à traiter d'une part les grammaires sous la forme de RTN pondérés (au lieu de règles de réécritures hors-contexte) et d'autre part pour accepter une entrée sous la forme d'un automate sur les mots étiquetés (au lieu d'une séquence de mots). Notre analyseur produit comme résultat, pour chaque phrase analysée, une forêt partagée d'arbres d'analyse pondérés ; cette forêt peut ensuite être passée à un module de traitement indépendant de notre analyseur permettant de produire différents types de résultats : concordances, texte annoté, automate du texte modifié, formulaires XML, etc. Nous terminons ainsi cette partie en présentant les différentes applications textuelles que nous avons implémentées sous la forme de modules qui sont branchés à notre analyseur.

3.2 Présentation générale du projet Outilex

Le projet Outilex regroupe 10 partenaires français, dont 4 académiques (Université de Marne-la-Vallée, Université de Rouen, le LIP6, et le LORIA) et 6 industriels (Systran, Thales Communication, Thales R&D, LCI, Lingway et le CEA). Le projet est coordonné par l'IGM et financé par le ministère de l'Industrie dans le cadre du Réseau national des technologies logicielles (RNTL). Préparé sous la direction de Maurice Gross, il a été lancé en 2002 et doit se terminer en 2006.

Les méthodes de traitement des langues naturelles sont encore aujourd'hui, la plupart du temps, mises en oeuvre par des logiciels dont la diffusion est limitée. De plus, on dispose rarement de formats d'échange ou de convertisseurs de formats qui permettraient de combiner plusieurs composants logiciels pour un même traitement. Quelques plates-formes font exception à cette situation générale, mais aucune n'est totalement satisfaisante. Intex [Silberztein, 1993], FSM [Mohri *et al.*, 1998] et Xelda[3] sont fermés au développement collaboratif. Unitex [Paumier, 2003b], inspiré d'Intex mais dont le code source est pour la quasi-totalité sous licence LGPL, ne fournit pas de formats XML. Les systèmes NLTK [Loper et Bird, 2002] et Gate [Cunningham, 2002] n'ont pas de fonctionnalités de gestion de ressources lexicales.

Outilex a donc pour objectif de combler ce manque en proposant des modules qui effectuent toutes les opérations fondamentales pour les traitements de texte écrit : traitements sans lexiques, exploitation des lexiques et des grammaires et gestion des ressources linguistiques. Les données manipulées à toutes les étapes du traitement sont structurées dans des formats XML compatibles avec les normes en cours de validation sur la représentation des ressources linguistiques, et également dans des formats binaires plus compactes permettant des traitements plus efficaces ; les convertisseurs entre ces formats sont fournis par la plate-forme.

Une interface graphiques programmée en Java intègre tous ces modules et fait appel à leurs fonctionnalités par l'intermédiaire de programmes écrits en C++. L'interface permet de travailler sur différents projets regroupant un

[3]http ://www.dcs.shef.ac.uk/ hamish/dalr/baslow/xelda.pdf.

ensemble de ressources (textes, dictionnaires et grammaires). L'utilisateur peut y définir sa propre chaîne de traitement et visualiser les entrées, les sorties et les résultats intermédiaires dans un conteneur à onglets dédié à cet effet.

3.2.1 Segmentation du texte

Le module de segmentation prend en entrée un texte brut ou HTML et il produit en sortie le texte segmenté en paragraphes, en phrases et en tokens dans un format XML (seg.xml) proche de celui proposé par le projet de norme ISO d'annotation morpho-syntaxique de textes (MAF) [Clément et de la Clergerie, 2005] élaboré dans le cadre du projet RNIL[4]. Nous n'avons pas inclus dans la plateforme de traitements applicatifs opérant sur le modèle de texte représenté comme une séquence de tokens, ni sur le modèle encore plus simple du sac de tokens, mais de tels traitements nous semblent faciles à interfacer, en raison justement de la simplicité des modèles sous-jacents.

Dans le cas où le texte fourni en entrée à notre segmenteur est sous la forme d'un document HTML, les balises HTML de mise en page sont conservées dans le résultat, placées dans des éléments XML qui les distinguent des données textuelles. De cette manière, nous ne perdons aucune donnée présente dans le document original. L'opération de segmentation est donc réversible et il est donc possible de reproduire à tout moment le document (éventuellement modifié lors de futurs traitements) dans sa mise en page d'origine.

Les règles de segmentation en tokens et en phrases sont basées sur la catégorisation des caractères définie par la norme Unicode. À chaque token est associé un certain nombre d'informations telles que son type (mot, nombre, ponctuation, etc.), son alphabet (latin, grec), sa casse (mot en minuscule, commençant par une majuscule, etc.) ainsi que d'autres informations pour les autres symboles (signe de ponctuation ouvrant ou fermant, etc.). De plus, un identifiant est associé à chaque token qui sera conservé durant toutes les phases du traitement. Par exemple, la phrase *La police a saisi 164 procès-verbaux jeudi dernier* est segmentée comme dans la figure 3.1.

[4]Ressources Normalisées en Ingénierie des Langues.

```
<?xml version="1.0"?>
<document original_format="txt"><par id="1"><tu id="s0"><token type="word"
id="t1" alph="latin" case="capit">La</token> <token type="word" id="t2"
alph="latin">police</token> <token type="word" id="t3" alph="latin">a</token>
<token type="word" id="t4" alph="latin">saisi</token> <token type="numeric"
id="t5">164</token>
[...]
<token type="punctuation" id="t11">.</token></tu></par>
</document>
```

FIG. 3.1 – Texte segmenté au format seg.xml

3.2.2 Traitement par lexiques

Les traitements évoqués dans la partie précédente ont pour résultat une représentation du texte comme séquence de tokens. Nous pensons qu'une plateforme généraliste doit intégrer certaines notions fondamentales absentes de ce modèle, comme celle de mots composés ou expressions multi-mots, ou la séparation des emplois en cas d'ambiguïté. Les produits de la linguistique de corpus seuls [Schmid, 1994] ne sont pas de nature à résoudre les problèmes posés par de telles notions. L'un des moyens pour y parvenir est l'utilisation de lexiques et de grammaires. L'utilisation de lexiques par les entreprises du domaine s'est d'ailleurs largement étendue au cours des dernières années. C'est pourquoi Outilex fournit un jeu complet de composants logiciels pour les opérations sur les lexiques. De plus, dans le cadre de sa contribution à Outilex, l'IGM a rendu publique une proportion substantielle des lexiques du LADL pour le français (109 912 lemmes simples et 86 337 lemmes composés) et l'anglais (166 150 lemmes simples et 13 361 lemmes composés). Le jeu d'étiquettes pour le français combine 13 catégories morpho-syntaxiques, 18 traits flexionnels et divers traits syntaxico-sémantiques. Ces ressources sont proposées sous la licence LGPL-LR, créée dans le cadre d'Outilex et agréée par la Free Software Foundation. Les programmes d'Outilex sont compatibles avec toutes les langues européennes à flexion par suffixes. Des extensions seront nécessaires pour les autres types de langues.

Formats des lexiques

L'absence de formats génériques et de documentation sur les données sont deux obstacles à l'utilisation et à l'échange de lexiques pour le traitement automatique des langues. Les formats adoptés par Outilex tirent parti de deux circonstances : d'une part, l'émergence actuelle de modèles de données consensuels dans les projets de normalisation ; d'autre part, le fait que l'IGM dispose de lexiques du français construits manuellement et d'une grande couverture[5]. Nous avons traduit en XML le format Dela et inséré dans les balises de la documentation sur les données. Le format obtenu pour les lexiques de formes fléchies, dic.xml, illustré par la figure 3.2, est adapté à l'échange de données et compatible avec le modèle LMF [Francopoulo, 2003]. Le format Dela, plus compact :

appelés du contingent,appelé du contingent.N+hum :mp

est adapté à la visualisation sur écran et à la maintenance manuelle par les linguistes [Laporte, 2005]. Nous avons donc réalisé des convertisseurs dans les deux sens entre ces deux formats. Il existe en outre un format opérationnel que nous décrivons dans la section 3.2.4.

Consultation des lexiques

L'étiqueteur morpho-syntaxique d'Outilex prend un texte segmenté au format *seg.xml* en entrée et attribue à chaque forme (simple ou composée) l'ensemble des étiquettes lui correspondant extraites des lexiques indexés (cf. section 3.2.4). Il est possible d'appliquer un ensemble de lexiques à un texte dans la même passe de traitement. De plus, un système de priorités permet de bloquer des analyses issues de lexiques à faible priorité si la forme considérée est également présente dans un lexique de priorité supérieure. Ainsi, nous fournissons par défaut un lexique général proposant un grand nombre

[5]Ces deux circonstances sont liées à des travaux effectués au LADL sous la direction de Maurice Gross, la première par l'intermédiaire du projet Genelex de normalisation de lexiques [Normier et Nossin, 1990], la deuxième à travers le système de lexiques Dela [Courtois, 1990, Courtois, 2004].

```
<entry>
  <lemma>appelé du contingent</lemma>
  <pos name='noun'/>
  <feat name='subcat' value='human'/>
  <inflected>
    <form>appelés du contingent</form>
    <feat name='gender' value='masculine'/>
    <feat name='number' value='plural'/>
  </inflected>
</entry>
```

FIG. 3.2 – Un extrait de lexique au format dic.xml

d'analyses pour la langue standard, que l'utilisateur peut, pour une application spécifique, enrichir à l'aide de lexiques complémentaires et/ou filtrer avec un lexique prioritaire.

Enfin, plusieurs options, qui peuvent se combiner entre elles, permettent de paramétrer la méthode de consultation afin de faire de la recherche de formes approchée. Il est ainsi possible :

– d'ignorer complètement la casse ;
– d'ignorer les caractères majuscules lorsqu'ils apparaîssent dans les textes mais pas lorsqu'ils apparaîssent dans les dictionnaires (utile pour la reconnaissance des noms propres) ;
– de ne pas prendre en compte durant la recherche la présence ou l'absence des accents et autres signes diachritiques.

Tous ces paramètres permettent d'adapter notre étiqueteur en fonction de la nature du texte analysé (article de journal, page internet, e-mail, etc.).

Représentation du texte étiqueté

L'utilisation exclusive de lexiques pour étiqueter les textes produit des ambiguïtés lexicales. Le modèle le plus adapté pour représenter le texte étiqueté dans ces conditions est l'automate fini acyclique, en général appelé "treillis de mots" dans ce contexte. La figure 3.3 présente une partie de l'automate du texte obtenu après l'étiquetage de la phrase segmentée présentée dans la section 3.2.1. Ce modèle prend en compte la notion de mot, distincte de la notion de token en raison notamment des expressions multi-mots.

La plate-forme Outilex a mis au point deux formats nouveaux pour la représentation du texte étiqueté. Le premier est le format binaire de sortie de l'outil de consultation des lexiques. Il permet un traitement rapide de textes de grande taille. Le deuxième, fsa.xml, est destiné à l'échange de données (figure 3.2.2). Le projet MAF propose un format voisin. Des fonctionnalités d'import/export entre ces deux formats sont prévues. Un convertisseur entre le format binaire et le format fsa.xml est disponible. De plus, les deux formats peuvent être exportés vers le format dot [Gansner et North, 2000].

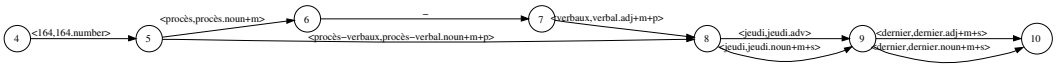

FIG. 3.3 – Automate du texte

3.2.3 Traitements utilisant des grammaires

Les traitements utilisant des grammaires prennent en entrée un texte étiqueté sous la forme d'un automate du texte et une grammaire WRTN. Ils identifient dans le texte les séquences concordantes avec la grammaire et produisent en sortie des résultats d'analyse sous différentes formes. Tous ces traitements reposent sur un même moteur d'analyse, que nous décrivons dans la section 3.5. Notre analyseur fournit comme résultat une forêt partagée d'arbres d'analyse pondérés pour chaque phrase analysée, les noeuds des arbres étant

```
<q id="31" pos="138">
 [...]
 <tr to="33">
  <lex>
   <form>procès</form>
   <lem>procès</lem>
   <pos v="noun"/>
   <f n="proper" v="false"/>
   <f n="gender" v="m"/>
  </lex>
 </tr>
```

```
<tr to="35">
 <lex>
  <form>procès-verbaux</form>
  <lem>procès-verbal</lem>
  <pos v="noun"/>
  <f n="proper" v="false"/>
  <f n="gender" v="m"/>
  <f n="number" v="p"/>
 </lex>
</tr>
</q>
```

FIG. 3.4 – Extrait d'un automate acyclique représentant un texte étiqueté

décorés par les éventuelles sorties présentes dans la grammaire. Cette forêt est ensuite passée, après l'analyse de chaque phrase, à un module de post-traitement permettant de produire différents types de résultat :

– une liste de concordances,
– un texte annoté,
– un automate du texte modifié,
– etc.

L'implémentation de notre analyseur ainsi que tous les différents traitements utilisant des grammaires que nous avons implémentés sont développés plus en détail dans la suite de ce chapitre.

3.2.4 Gestion de ressources linguistiques

La réutilisation de lexiques présuppose une certaine flexibilité. Un lexique n'est pas une ressource statique : en raison de l'évolution de la langue, et en particulier de la langue technique, des mises à jour régulières sont nécessaires. La gestion des lexiques et grammaires repose sur deux points : construction et maintenance manuelles des ressources dans un format li-

sible ; compilation en un format directement opérationnel. Cependant, les manuels de traitement des langues naturelles, même généralistes et réputés, comme [Jurafsky et Martin, 2000], ne traitent pas ces techniques, qui nécessitent une collaboration étroite d'informaticiens et de linguistes ; et peu de systèmes fournissent les fonctionnalités requises (Xelda, Intex, Unitex). La plate-forme Outilex propose donc un jeu complet d'outils de gestion de ressources linguistiques.

Flexion automatique des lexiques

Le module de flexion automatique prend en entrée un lexique de lemmes et des règles de flexion au format XML ; il produit en sortie un lexique de formes fléchies. Par exemple, la figure 3.5 présente les rêgles de flexion pour les verbes anglais de type *to carry*. Le module produit entre autres pour ce verbe la forme fléchie *carries* associée au code flexionnel *prs_ 3s* (troisième personne du singulier au présent).

```
<Paradigm code="4">
   <StemTrigger>y</StemTrigger>
   <Inflection caseId="inf"><Form>y</Form></Inflection>
   <Inflection caseId="prs_1s"><Form>y</Form></Inflection>
   <Inflection caseId="prs_2s"><Form>y</Form></Inflection>
   <Inflection caseId="prs_3s"><Form>ies</Form></Inflection>
   <Inflection caseId="prs_p"><Form>y</Form></Inflection>
   <Inflection caseId="prt"><Form>ied</Form></Inflection>
   <Inflection caseId="imp"><Form>y</Form></Inflection>
   <Inflection caseId="ppt"><Form>ied</Form></Inflection>
   <Inflection caseId="ppr"><Form>ying</Form></Inflection>
</Paradigm>
```

FIG. 3.5 – Règle de flexion

Indexation des lexiques

Afin d'accélérer leur consultation (cf. 3.2), les lexiques sont indexés sur les formes fléchies en utilisant une représentation par automate fini minimal

[Revuz, 1991] qui permet de les comprimer tout en offrant un accès rapide à l'information. Le format binaire obtenu (fichier .idx) est adapté aux lexiques à jeu d'étiquettes riche. Le tableau de la figure 3.6 décrit l'indexation du DELAF français[Courtois, 1990] et la taille du fichier obtenu après l'indexation du même lexique par les outils équivalents délivrés avec Unitex.

# formes fléchies	# formes canoniques	taille DELA (utf8)	taille XML (xml.gz)	taille idx	temps d'indexation	taille Unitex
1264170	122035	35 Mo	220 Mo (5.3 Mo)	9.5 Mo	1m03s	59 Mo

FIG. 3.6 – Résultat de l'indexation du DELAF français

3.2.5 Perspectives de la plate-forme

La plate-forme Outilex, dans sa version préliminaire actuelle, effectue toutes les opérations fondamentales du traitement automatique du texte écrit : traitements sans lexiques, exploitation de lexiques et de grammaires, gestion de ressources linguistiques. Les données manipulées sont structurées dans des formats XML, et également dans d'autres formats plus compacts, soit lisibles soit binaires ; les convertisseurs de formats nécessaires sont inclus dans la plate-forme. Enfin, des lexiques issus du LADL, construits manuellement et d'une couverture substantielle, seront distribués avec la plate-forme sous licence LGPL-LR.

Le développement de la plate-forme a nécessité une expertise conjointe des éléments informatiques et linguistiques du problème ; il a pris en compte les besoins de la recherche fondamentale et ceux des applications. Au-delà de la fin du projet, l'avenir de la plate-forme Outilex est conçu dans le cadre du développement collaboratif. Nous espérons que la plate-forme actuelle, déjà compatible avec de nombreuses langues, sera étendue à d'autres et enrichies en fonctionnalités nouvelles.

3.3 Formalisme des WRTN

Le formalisme des grammaires WRTN utilisées dans Outilex appartient à la famille des réseaux de transitions récursifs (RTN) [Woods, 1970]. Une grammaire se présente ainsi comme une collection d'automates lexicaux récursifs. Chaque automate possède un nom unique ; leurs transitions sont étiquetées en entrée soit par un masque lexical (symbole terminal), soit par le nom d'un automate de la grammaire (symbole non terminal). Les transitions des automates peuvent également comporter des sorties sous la forme de chaînes de caractères.

Nous avons ajouté à ce modèle la notion de pondération : chaque transition est pondérée par un nombre réel. Ainsi, notre analyseur associe à chaque analyse obtenue un score correspondant à la somme des poids associés aux transitions de la grammaire qui ont permis d'obtenir cette analyse. Ce système permet de filtrer les résultats de notre analyseur, dans le cas où une grammaire ambiguë fournirait plusieurs analyses candidates pour la même séquence, en ne conservant que celles ayant le score le plus élevé. Les poids sur les transitions peuvent être assignés manuellement par le grammairien à partir de considérations linguistiques (ou de façon adhoc à partir d'observations empiriques). Ce système de pondération sur les transitions rend également possible la réalisation de systèmes hybrides utilisant à la fois des méthodes statistiques et des méthodes fondées sur des ressources linguistiques ; le poids qui étiquettent les chemins dans les grammaires pouvant par exemple être calculés par entraînement des grammaires sur des corpus d'apprentissages[Nasr, 2004].

Les figures 3.7 et 3.8 présentent un exemple de grammaire WRTN. Les symboles dans les boîtes grisées sont des symboles non terminaux correspondant à des appels à des sous-graphes. Dans le graphe principal, le poids de 1 dans le chemin correspondant à l'analyse avec expression figée (indiqué par la sortie `Vfigé/1`) rend cette analyse prioritaire par rapport l'analyse compositionnelle décrite dans l'autre chemin (de poids 0 par défaut).

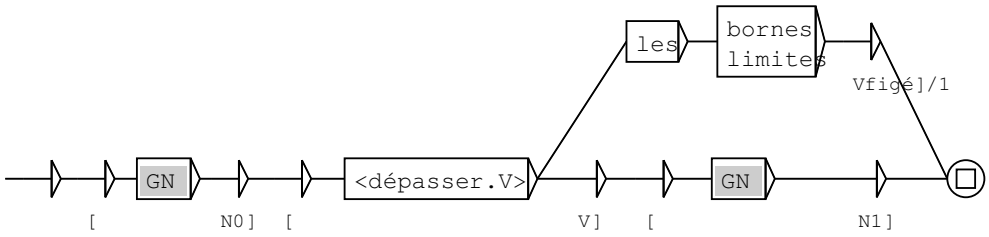

FIG. 3.7 – Exemple de grammaire WRTN : graphe principal

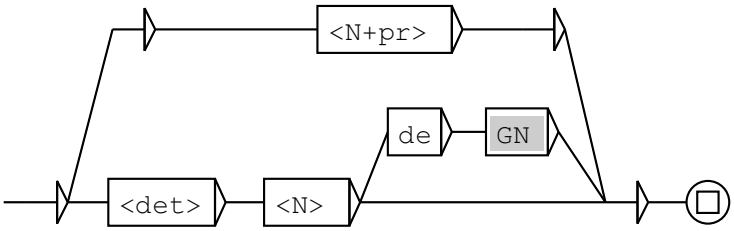

FIG. 3.8 – Sous-graphe GN

Ainsi par exemple, le résultat de l'application de cette grammaire (de façon non glissante) sur les phrases suivantes :

Luc dépasse les limites.
Luc dépasse les limites de son art.
Luc dépasse le cap de Beauregard.

donnera les analyses :

[Luc N0] [dépasse les limites Vfigé].
[Luc N0] [dépasse V] [les limites de son art N1].
[Luc N0] [dépasse V] [le cap de Beauregard N1].

La première analyse ayant un score de 1 et les deux suivantes un score de 0.

3.3.1 Définition formelle

Formellement, étant donné un jeu d'étiquettes T, une grammaire WRTN est définie comme étant un tuple $G = (\mathcal{N}, Lex, \mathcal{A}, \theta, S)$ où

- $\mathcal{N}$ est l'ensemble des symboles non terminaux, ce sont les identifiants des automates de la grammaire ;
- Lex est l'ensemble des masques lexicaux définis sur le jeu d'étiquettes T ;
- $\mathcal{A}$ est l'ensemble des automates pondérés récursifs dont le formalisme est décrit plus en détail par la suite ;
- $\theta : \mathcal{N} \mapsto \mathcal{A}$ est une fonction bijective qui associe à tout symbole non terminal de la grammaire un automate de $\mathcal{A}$;
- $S \in \mathcal{N}$ est le symbole non terminal axiome de la grammaire.

Un automate A d'une grammaire WRTN G se définit comme un tuple $A = (Q, q_0, F, Lex, N, \Sigma, \Delta)$ où

- Q est un ensemble fini d'états ;
- $q_0 \in Q$ est l'état initial ;
- $F \subseteq Q$ est l'ensemble des états finaux ;
- Lex est l'ensemble des masques lexicaux définis sur le jeu d'étiquettes donné ;
- $\mathcal{N}$, l'ensemble des symboles non terminaux de G ;
- Σ, l'alphabet de sortie (dans le cadre de nos applications il s'agit de l'ensemble des caractères Unicode) ;
- $\Delta \subseteq Q \times (N \cup Lex) \times \Sigma^* \times \mathcal{R} \times Q$ un ensemble de transitions.

Les automates que nous manipulons dans nos grammaires WRTN ont toujours un unique état initial. Etant donné un automate A, on dénotera par $init(A)$ son état initial.

Une transition d'un automate d'une grammaire WRTN est définie comme un 5-tuple $t = (q_o, \alpha, \beta, w, q_d)$ où $q_o \in Q$ est l'état de départ, $\alpha \in (N \cup Lex)$ le symbole d'entrée, $\beta \in \Sigma^\star$ la chaîne en sortie, $w \in \Re$ le poids associé à la transition et $q_d \in Q$ son état de destination. Par la suite, étant donnée une transition t, nous dénoterons par $orig(t)$, $label(t)$, $out(t)$, $weight(t)$ et $dest(t)$ ces cinq éléments respectifs.

Afin de mieux distinguer les transitions étiquetées par des symboles terminaux des transitions étiquetées par des symboles non-terminaux, nous appelons désormais *transitions lexicales*, les transitions étiquetées par des masques lexicaux et *transitions syntaxiques*, les transitions étiquetées par des symboles non terminaux.

3.4 Optimisation des grammaires

Avant d'être utilisées pour l'analyse de textes, nous précédons préalablement à des optimisations sur les grammaires WRTN. Au cours de cette opération, chaque automate de la grammaire est optimisé par les opérations d'émondation, de suppression des transitions étiquetées par le mot vide, de déterminisation et de minimisation, telles que nous les avons définies sur les automates lexicaux (cf. chapitre 1). En ce qui concerne la déterminisation des automates, nous n'appliquons pas notre algorithme de déterminisation des transducteurs lexicaux (cf. section 2.8) dans lequel nous procédons à un décalage des sorties de la grammaire. En effet, comme on l'a vu, afin de pouvoir traiter tous types de transducteurs nous avons dû procéder à la déterminisation locale de ces grammaires durant l'analyse, ce qui complique l'implémentation de notre analyseur. De plus, même si tous les automates qui composent une grammaire WRTN sont déterministes, la grammaire peut ne pas pour autant être déterministe, du fait notamment que plusieurs sous-graphes qui sont appelés à partir d'un même état peuvent avoir des préfixes communs. Dans [Paumier, 2003a], S. Paumier propose une méthode pour limiter ce genre de non déterminisme dans les grammaires RTN. Cette méthode consiste en une transformation de la grammaire inspirée de la mise en forme normale de Greibach[Greibach, 1965] utilisée pour les grammaires hors-contextes. Nous n'avons pas retenu cette méthode non plus, car si la grammaire ainsi optimisée reconnaît bien le même langage que la grammaire originale, en revanche la transformation de la grammaire est telle que les arbres de dérivations obtenus comme résultat de l'analyse d'une même séquence diffèrent. De plus, cette opération ne permet que d'optimiser localement la grammaire qui peut rester encore non déterministe dans sa globalité [6]. Ainsi, plutôt que de tenter

[6]D'une manière générale, il existe des langages algébriques tels que toutes les grammaires qui les engendrent sont nécessairement ambiguës ; le langage $\{a^n b^m c^p, \text{avec } n =$

d'optimiser les grammaires afin de limiter au maximum leur non déterminisme qui ne pourra de toute façon pas être totalement éliminé dans le cas général, nous avons préféré concentrer notre travail sur l'implémentation d'un algorithme d'analyse qui puisse traiter efficacement les grammaires ambiguës en utilisant des techniques de programmation dynamique permettant d'explorer toutes les possibilités d'analyses sans avoir recours à des procédures coûteuses de *backtracking*.

Nous avons également implémenté une procédure d'applatissement controlé des grammaires WRTN, en reprenant les principes décrits dans [Paumier, 2003a]. Cette opération transforme une grammaire WRTN en un transducteur lexical fini équivalent, éventuellement à une approximation près, en faisant remonter les sous-graphes dans le graphe principal. Dans le cas où la grammaire contient une chaîne d'appels à des sous-graphes récursive, un paramêtre spécifiant la profondeur d'exploration maximale permet d'interrompre l'applatissement de la grammaire et d'obtenir un transducteur qui n'est pas équivalent à la grammaire d'origine. Le transducteur obtenu occupe plus d'espace mémoire dans le cas général mais son utilisation peut accélérer sensiblement les traitements.

3.5 Analyse

La majorité des algorithmes d'analyse par grammaire algébriques utilisés en TAL sont des *analyseurs tabulaires* qui reposent sur les algorithmes CYK [Younger, 1967] ou de Earley [Earley, 1970] ou des dérivés. Ces algorithmes utilisent des techniques de programmation dynamique[Bellman, 1957] ce qui les rend particulièrement bien adaptés pour traiter les grammaires ambiguës qui sont très courantes en traitement des langues.

Notre algorithme d'analyse que nous présentons dans cette section est un analyseur à charte descendant inspiré de l'algorithme de Earley. Les principales différences avec ce dernier est que nous traitons des grammaires WRTN sous la forme d'un réseau d'automates pondérés (au lieu d'une grammaire algébrique hors-contexte) et que nous prenons un texte en entrée sous la forme

m ou $m = p\}$ en est un exemple.

d'un automate acyclique (à la place d'une séquence de mots).

D'autre part, l'algorithme d'Earley est généralement présenté dans la littérature comme un simple reconnaisseur, c'est-à-dire qu'étant donné une chaîne et une grammaire en entrée, l'algorithme détermine si la chaîne appartient au langage engendré par la grammaire. Ici, nous présentons notre algorithme comme un analyseur, et nous explicitons les procédures qui permettent de produire en sortie une analyse sous la forme d'une forêt partagée d'arbres syntaxiques annotés et pondérés.

3.5.1 Algorithme

L'algorithme fonctionne à l'aide d'un tableau dans lequel sont stockées toutes les analyses partielles en cours de traitement. L'utilisation de cette table est la clef de l'efficacité de notre algorithme puisqu'elle permet, en conservant tous les résultats partiels, de ne pas recalculer plusieurs fois les mêmes sous-analyses. La table (notée $chart[]$) a la taille de l'ensemble d'états de l'automate de phrase analysé. A chaque position, se trouve un ensemble des analyses en cours se terminant à la position correspondante dans l'automate du texte. Une analyse en cours correspond à la reconnaissance partielle d'un segment du texte par un automate de la grammaire ; elle contient les informations telles que l'état de départ dans l'automate du texte à partir duquel la reconnaissance a débuté, les états courants de l'analyse dans l'automate du texte et dans l'automate de la grammaire. Nous conservons également dans une analyse en cours les traces du chemin parcouru dans le texte qui a permis la reconnaissance, ainsi que les sorties qui ont été produites lors du franchissement des transitions de la grammaire et le score courant de l'analyse (qui consiste en la somme des poids des transitions franchies dans la grammaire).

Formellement, nous définissons une analyse en cours comme un tuple $a = (A, q, i, j, p, o, w)$ où

- A est l'automate de la grammaire G qui décrit la séquence en cours de reconnaissance par l'analyse en cours ;
- q est un état de A, c'est la position courante dans A de l'analyse en cours ;
- i et j sont des états de l'automate du texte, ce sont les positions de début

et de fin de la séquence reconnue par l'analyse en cours ;
– p est le chemin de l'analyse en cours, c'est une séquence composée de transitions lexicales de l'automate du texte et de transitions syntaxiques qui correspondent à des analyses en cours complétées ;
– o représente les sorties de la grammaire qui ont été émises durant l'analyse en cours, c'est une séquence de sorties (sous la forme de chaînes de caractères) de la même taille que p, qui correspond à toutes les sorties (éventuellement sous la forme de chaînes vides) associées aux transitions de A qui ont permis la reconnaissance pour l'analyse en cours ;
– w est le score associé à l'analyse en cours.

Nous distinguons deux types particuliers d'analyse en cours : les *analyses vierges* et les *analyses complétées*.

Un *analyse vierge* est une analyse en cours qui est créée en préambule à toute tentative de reconnaissance d'une séquence par un automate A de la grammaire à partir d'une position i donnée dans l'automate du texte. Une telle analyse a la forme $a = (A, init(A), i, i, \epsilon, \epsilon, 0)$.

Une *analyse complétée* (ou *analyse réussie*) est une analyse en cours qui a effectivement abouti à la reconnaissance d'un segment du texte par un automate de la grammaire. Une analyse en cours $a = (A, q, i, j, p, o, w)$ est complétée ssi q est un état final de A.

Notre algorithme général d'analyse consiste à avancer séquentiellement dans le texte en parcourant l'automate du texte état par état ; à chaque position, nous parcourons la pile des analyses en cours et nous empilons de nouvelles analyses lors du franchissement de transitions lexicales et syntaxiques dans la grammaire. Nous détaillons toutes ces étapes dans les sections qui suivent.

Empilement d'une analyse en cours

La procédure ENQUEUE est appelée pour empiler une nouvelle analyse en cours dans la charte. La procédure prend en entrée une analyse en cours a, et une position i dans l'automate du texte. Elle rajoute cette analyse dans la charte à la position q si elle n'y est pas déjà présente.

entrée:
a, une analyse en cours
i, un état de l'automate du texte
si a n'est pas présent dans $chart[i]$ **alors**
PUSH($chart[i], a$)
fin si

Algorithme 6: procédure ENQUEUE

Dans le cas où deux analyses en cours recouvrent le même segment du texte (c'est-à-dire si elles ont les mêmes états de départ et d'arrivée), et si elles ont une position identique dans la grammaire (même automate et même état), c'est l'analyse de poids le plus élevé qui est conservée. Nous avons également ajouté la possibilité de ne conserver qu'une seule analyse, si les deux analyses ont un score identique. Cette option permet d'accélérer sensiblement les traitements lors de l'application de grammaires fortement ambiguës, en contrepartie lorsqu'elle est activée, elle peut être la cause de silence puisque l'analyseur ne produira alors au plus qu'une analyse pour un segment du texte donné, contrairement au cas général où l'analyseur peut produire plusieurs analyses qui diffèrent par leurs arbres de dérivations et/ou les sorties émises par la grammaire. Cette fonctionnalité reste néanmoins utile pour certaines applications pour lesquelles on ne souhaite pas récupérer plus d'une analyse pour un même segment du texte (par exemple, pour la génération de textes annotés présentée dans la section 3.6.3).

Prédiction d'une nouvelle sous-analyse

La procédure PREDICTOR est appelée lorsque l'on cherche à reconnaître un symbole non terminal X à partir d'une position i de l'automate du texte. La procédure consiste à empiler dans la table à la position i une nouvelle analyse vierge correspondant à l'automate X de la grammaire.

entrée:
i, un état dans l'automate de phrase
A, un automate de la grammaire

ENQUEUE$(q, (A, init(A), i, i, \epsilon, \epsilon, 0))$

Algorithme 7: procédure PREDICTOR

Franchissement d'une transition lexicale

La procédure SCANNER est appelée lors du franchissement d'une transition lexicale dans la grammaire. Étant donné une analyse en cours $a = (A, q, i, j, p, o, w)$ et une transition lexicale t sortant de l'état q de A, on parcourt les transitions sortantes de l'état j de l'automate du texte T et pour chaque transition t_2 dont l'étiquette concorde avec l'étiquette de t, on empile dans la table, à la position de l'état d'arrivée de t_2, une nouvelle analyse en cours correspondant au franchissement de ces deux transitions.

entrée:
$a = (A, q, i, j, p, o, w)$ une analyse en cours
$t = (q, m, o', w', q')$ une transition lexicale de A sortant de l'état q

1:
2: **pour tout** transition $t_2 = (j, m', k)$ sortant de l'état j de T **faire**
3: **si** INTERSECT(m, m') **alors**
4: ENQUEUE$(k, (A, q', i, k, p \cdot t_2, o \cdot o', w + w'))$
5: **fin si**
6: **fin pour**

Algorithme 8: procédure SCANNER

Complétion d'une sous-analyse

La procédure COMPLETER est appelée lorsque la reconnaissance d'une séquence par un automate A a réussi : on est arrivé à un état final de cet automate dans l'analyse en cours.

Pour chaque analyse en cours présente dans la table à la position de l'origine du segment reconnu, si une transition sortante de la grammaire est étiquetée par l'automate A, on ajoute une nouvelle analyse en cours correspondant au franchissement de cette transition syntaxique dans la table à la position de la fin du segment reconnu.

entrée:
$a = (A, q, i, j, p, o, w)$ une analyse en cours réussie (q est un état final de A)

1:
2: **pour tout** analyse en cours $b = (A', q', i', i, p', o', w')$ dans $chart[i]$ **faire**
3: **pour tout** transition $t = (q', X, o'', w'', q'')$ sortante de l'état q' de A' **faire**
4: **si** X est un symbole non terminal et $\theta(X) = A$ **alors**
5: $c \leftarrow (A', q'', i', j, p' \cdot a, o' \cdot o'', w + w' + w'')$
6: ENQUEUE(j, c)
7: **fin si**
8: **fin pour**
9: **fin pour**

Algorithme 9: procédure COMPLETER

La nouvelle analyse en cours c est construite à la ligne 5. Celle-ci consiste en une avancée de l'analyse en cours b par le franchissement d'une transition syntaxique concordant avec l'analyse complétée a. Ainsi, le chemin de la nouvelle analyse en cours c consiste en la concaténation du chemin de b avec l'analyse complétée a ($p' \cdot a$). De même, les sorties associées à c consiste en la concaténation des sorties associées à b avec la sortie associée à la transition t étiquetée en entrée par le symbole non terminal ($o' \cdot o''$). Enfin, le score associé à cette nouvelle analyse en cours est la somme du score associé à l'analyse b, plus celui associé à l'analyse complétée plus celui associé à la transition t ($w + w' + w''$).

Initialisation

La procédure d'initialisation est appelée en préambule à l'analyse. Elle consiste à créer une table vide de la taille de l'automate de phrase à analyser, puis à empiler à la position 0 de la table une analyse vierge associée à l'axiome de la grammaire si l'on souhaite faire une analyse complète de la phrase. Dans le cas où l'on souhaite appliquer la grammaire de façon glissante sur le texte, c'est-à-dire si l'on souhaite trouver toutes les concordances de la grammaire à partir de chaque position dans le texte, on initialise chaque pile de la table par une analyse vierge dont l'automate est l'axiome de la grammaire.

entrée:
$G = (\mathcal{N}, Lex, \mathcal{A}, \theta, S)$ une grammaire WRTN
$T = (Q, q_0, F, Lex, \delta)$ un automate de phrase
surf un booléen qui spécifie si la grammaire doit être appliquée de façon glissante

1: *chart* $\leftarrow$ une charte vierge de taille $card(Q)$
2: **si** *surf* = VRAI **alors**
3: **pour tout** état q de T **faire**
4: PREDICTOR(q, S)
5: **fin pour**
6: **sinon**
7: PREDICTOR(q_0, S)
8: **fin si**

Algorithme 10: Initialisation

Analyse

La fonction principale de notre analyseur est présentée dans la figure 11. Elle consiste à parcourir séquentiellement toutes les positions dans l'automate du texte (ligne 1). L'automate du texte en entrée doit avoir été préalablement trié topologiquement. Cette propriété est essentielle pour la correction de l'algorithme puisqu'elle permet de s'assurer que lorsqu'on arrive à une position q, tous les états permettant d'accéder à q ont déjà été traités. Pour chacune

de ces positions, nous parcourons la pile des analyses en cours se terminant à cette position (ligne 2). Si l'analyse courante est complétée, on appelle la procédure COMPLETER (ligne 4), qui va permettre de faire avancer des analyses en cours par franchissement d'une transition syntaxique et de les rajouter à la pile courante. Ensuite, nous parcourons les transitions sortantes dans la grammaire pour l'analyse courante (ligne 6). Si la transition est étiquetée par un symbole non-terminal, on appelle la procédure PREDICTOR (ligne 8) qui empile une analyse vierge sur la pile des analyses en cours à la position courante. Si la transition est étiquetée par un symbole terminal, on appelle la procédure SCANNER (ligne 10) qui va tenter de consommer un symbole en entrée en franchissant une transition lexicale, et empiler une nouvelle analyse en cours à l'état d'arrivée de cette transition lexicale.

entrée:
$G = (N, Lex, \mathcal{A}, \theta, S)$ une grammaire WRTN
$T = (Q, q_0, F, Lex, \delta)$ un automate de phrase

1: **pour tout** état j de T **faire**
2: **pour tout** analyse en cours $a = (A, q, i, j, p, o, w)$ de $chart[j]$ **faire**
3: **si** q est final **alors**
4: COMPLETER(a)
5: **fin si**
6: **pour tout** transition $t = (q, X, o', w', q')$ sortant de q dans A **faire**
7: **si** X est un symbole non terminal **alors**
8: PREDICTOR(j, X)
9: **sinon**
10: SCANNER(a, t)
11: **fin si**
12: **fin pour**
13: **fin pour**
14: **fin pour**

Algorithme 11: Algorithme d'analyse

Une fois l'algorithme terminé, il suffit de regarder à la dernière position de la table, s'il existe une analyse complètée dont l'automate est l'axiome de la grammaire et qui recouvre la totalité de la phrase pour savoir si la phrase a été correctement analysée dans sa totalité. De même, dans le cas d'une

application glissante de la grammaire, toutes les occurrences dans la table d'analyses complètées de l'axiome de la grammaire décrivent les différentes analyses de tous les segments du texte concordant avec la grammaire.

3.5.2 Création de la forêt partagée d'arbres d'analyse

Le moteur de notre analyseur décrit dans la section précédente produit comme résultat une table qui contient l'ensemble des analyses en cours (complétées ou non) qui ont été traitées durant l'analyse. Il nous reste donc à construire la forêt d'arbres d'analyse en extrayant de cette table l'ensemble des analyses qui ont effectivement abouti à la reconnaissance par la grammaire de la totalité de la phrase (ou d'un de ses segments dans le cas d'une application glissante).

La construction de la forêt se fait en deux passes. Nous parcourons une première fois la table et pour chaque analyse complétée correspondant à l'axiome de la grammaire, nous notons par un drapeau cette analyse et récursivement les sous-analyses dépendantes de celle-ci. Dans une seconde passe, nous créons pour chaque analyse complétée ainsi annotée, un nouveau noeud dans notre forêt d'arbre d'analyse. Chaque noeud est décoré par des informations héritées de l'analyse qui lui correspond telles que le nom de l'automate, les sorties de la grammaire qui lui sont associées ou son score. Le chemin associé à l'analyse en cours permet de reconstruire la descendance du noeud : les transitions lexicales correspondant à des feuilles de l'arbre et les transitions syntaxiques à des noeuds internes de la forêt créés à partir d'autres analyses complétées. La forêt est bien partagée puisque chaque analyse en cours donne lieu à la création d'au plus un noeud interne, et ce indépendamment du nombre d'arbres de dérivation auxquels elle appartient.

3.6 Applications

Toutes les applications par grammaires que nous présentons ici utilisent le moteur d'analyse décrit dans la section précédente. Plus précisément, elles

ont été implémentées sous la forme de modules qui sont branchés à notre analyseur. Au niveau de l'API C++, la routine d'analyse qui prend en entrée une grammaire et un automate du texte et qui produit la forêt d'arbres d'analyses accepte un troisième argument sous la forme d'un *functor* C++, que l'on peut considérer comme une procédure externe de *post-traitement*. Cette procédure est appelée par notre analyseur avec comme paramètre la forêt d'arbres de dérivation obtenue après l'analyse de chaque phrase du corpus en cours de traitement, et peut produire tout type de résultats.

De cette manière nous séparons clairement les routines d'application d'une grammaire sur un texte et les routines qui permettent de formater les résultats. Cette architecture modulaire rend très aisé le développement de différents types d'applications utilisant des grammaires WRTN, puisqu'il suffit d'implémenter la routine qui se charge de transformer le résultat de l'analyse dans le résultat escompté et de brancher cette procédure de post-traitement sur notre analyseur que l'on a pas besoin de retoucher.

De plus, malgré la séparation du moteur d'analyse de la routine de post-traitement, cette dernière n'est pas invoquée après l'analyse du texte dans sa totalité, mais est appelée par notre analyseur après l'analyse de chaque phrase. Ainsi, lors de leurs traitements, les corpus sont lus séquentiellement phrase par phrase ; chaque phrase est d'abord analysée par la grammaire, puis la routine est appelée et les structures de données sont ensuite libérées avant de passer à l'analyse de la phrase suivante. Ce mode de fonctionnement est intéressant, notamment pour le traitement de corpus de textes de grande taille, puisqu'il nous assure que la consommation mémoire du programme ne dépend pas de la taille du texte traité, mais simplement de la longueur de chaque phrase (et de la taille des structures produites lors de leurs analyses).

Enfin, l'analyseur n'impose aucune contrainte sur la forme des sorties qui étiquettent la grammaire. Celles-ci sont considérées comme de simples chaînes de caractères et sont reproduites telles quelles dans les arbres de dérivation résultant de l'analyse. Cependant il n'en est pas de même pour la routine de post-traitement, qui peut donner une interprétation sémantique au contenu des sorties et ainsi leur attacher des actions particulières. Ceci offre la possibilité à tous types de traitements qui dépassent largement la simple transduction de chaînes de caractères. Nous avons ainsi pu implémenter, à partir du même moteur d'analyse général, diverses applications qui, à partir de gram-

maires WRTN, produisent différents types de résultats tels que la création de textes annotés, le modification de l'automate du texte ou encore la génération de formulaires dans le contexte de l'extraction d'informations. Toutes ces applications sont détaillées dans les sections suivantes.

3.6.1 Sérialisation de la forêt

Nous avons développé un module permettant de sérialiser dans un document XML la forêt, résultat de l'application d'une grammaire sur un texte. Cette forêt présente l'inconvénient d'avoir une grosse taille (de l'ordre de dix fois la taille de l'automate du texte). Cette fonctionnalité reste néanmoins utile lors de l'application de grammaires complexes, pour lesquelles le temps passé durant l'analyse grammaticale est supérieur au temps passé durant les traitements de la forêt d'arbres d'analyse ou encore lorsque l'on souhaite effectuer différents traitements sur cette forêt. La sauvegarde de la forêt sur le disque permet alors de lancer différentes routines de traitements sur celle-ci, en ne faisant l'analyse du texte qu'une seule fois. Notons que nous avons également implémenté une routine de post-traitement prenant en argument plusieurs routines de post-traitement, et qui, lorsqu'elle est appelée, déclenche les différentes routines qui lui sont passées en argument. Cette technique nécessite néanmoins de compiler un programme différent pour chaque combinaison de traitements que l'on souhaite appliquer au texte. La technique passant par la sauvegarde du résultat de l'analyse grammaticale dans un fichier intermédiaire ne souffre pas de cette limitation.

3.6.2 Concordancier

Comme première application, nous avons développé un concordancier qui permet de lister dans leur contexte d'apparition les différentes occurrences des motifs décrits par la grammaire. Le concordancier est avant tout une aide précieuse aux linguistes recherchant dans les textes des attestations de formes décrites dans des grammaires locales. Cet outil est également très pratique comme aide à l'écriture des grammaires puisqu'il permet de tester facilement leurs couvertures lorsque celles-ci sont en construction. Pour la première fois,

un système tel qu'INTEX ou Unitex permet d'exploiter les résultats de la levée d'ambiguïtés dans la construction des concordances.

La taille des contextes gauche et droit peut être paramétrée par l'utilisateur et les concordances peuvent être classées soit suivant leur ordre d'apparition dans le texte, soit par ordre lexicographique. De plus, il est possible de paramétrer la mise en forme des concordances, de manière à visualiser ou non les sorties de la grammaire, l'arbre de dérivation (sous forme parenthésée) ainsi que les scores des concordances. Notre concordancier construit les concordances et formate toutes ces informations dans un fichier XML ; une feuille de style XSLT permet ensuite de mettre en page ce document dans une page HTML en fonction des paramètres de visualisation. La figure 3.9 présente un extrait de l'index des concordances au format XML obtenu par l'application de notre grammaire d'extraction d'informations biographiques (cf. section 3.6.6) sur un corpus de textes en anglais. La figure 3.10 présente le même résultat mis en forme avec visualisation des sorties.

```
<match>
<left></left>
<center>
 <P n="Person_Phrases" w="0">
  <P n="position_phrases" w="0">
   <o>[job</o>
   <P n="TheJobIsNhum" w="0">
   <l f="The">&lt;The,the.lex+cap&gt;</l>
   <P n="position" w="0">
   [...]
  <o>job]</o>
 </P>
</P>
</center>
<right>. </right>
</match>
```

FIG. 3.9 – Concordances au format XML

FIG. 3.10 – Mise en forme des concordances

3.6.3 Création d'un texte annoté

Nous avons également développé la fonctionnalité d'application d'un transducteur sur le texte qui est proposée dans les systèmes INTEX et Unitex. Cette fonctionnalité produit en sortie un texte brut dans lequel sont insérées les sorties spécifiées dans la grammaire. Ces sorties peuvent au choix être insérées dans le texte d'origine ou remplacer les segments concordant avec la grammaire.

Ce procédé n'est pas complètement trivial puisque, d'une part, il nécessite de projeter la structure sous forme de graphe de l'automate du texte vers la structure linéaire du texte brut produit en sortie. D'autre part, il est également nécessaire de traiter le problème des chevauchements potentiels des segments du texte concordant avec la grammaire.

Notre algorithme consiste à traverser l'automate de phrase état par état en commençant par l'état initial. Pour chaque état, s'il n'existe pas de segment concordant commençant à cette position dans la forêt, on franchit la transition étiquetée par la catégorie LEX et on continue l'algorithme à partir de l'état d'arrivée de cette transition. S'il existe plusieurs segments concordant débutant à cette position du texte, alors nous sélectionnons la *meilleure* d'entre elles. Pour ce faire, nous filtrons ces concordances en appliquant successivement les trois règles de sélection suivantes, jusqu'à ce qu'il n'en reste plus qu'une seule :

- *Conserver les segments concordant qui recouvrent le segment du texte le plus long.* L'automate du texte étant trié topologiquement, il suffit de conserver les concordances dont l'état de destination est le plus élevé.
- *Conserver les concordances qui franchissent le moins de transitions lexicales.* A cette étape, toutes les concordances recouvrent le même segment du texte. Ainsi, si une concordance franchit un nombre moins élevé de transitions lexicales, c'est qu'elle comporte plus de transitions qui recouvrent plusieurs tokens.
- *Choisir une transition au hasard.*

La première heuristique permet d'appliquer le principe du *longest-match*. La seconde heuristique permet de préférer les analyses figées (mots composés)

aux analyses compositionnelles. Enfin, la dernière heuristique permet de ne conserver qu'une seule analyse. Notons que nous n'utilisons pas les poids associés aux analyses pour sélectionner la meilleure concordance, ceci est dû au fait que la sélection par le poids est déjà faite en amont au moment de l'analyse. Ainsi, lorsque la grammaire décrit plusieurs analyses recouvrant le même segment du texte, seules celles ayant le score le plus élevé apparaissent à l'issue de l'analyse. Ce système de pondération permet ainsi au grammairien d'influer sur le choix des concordances sélectionnées en pondérant de manière appropriée ces grammaires.

Une fois la concordance sélectionnée, nous produisons en sortie le résultat de la transduction de la grammaire par un parcours récursif de l'arbre de dérivation décoré par les sorties. Puis nous continuons le parcours à partir de l'état correspondant à la fin de la concordance. L'algorithme prend fin lorsque nous atteignons l'état final de l'automate de phrase.

3.6.4 Transduction sur l'automate du texte

La fonctionnalité d'application d'un transducteur à un texte telle que présentée dans la section précédente peut s'avérer très utile dans différents domaines du traitement des langues, telles que la reconnaissance d'entités, l'extraction d'informations ou encore l'analyse syntaxique. De ce fait, elle a été à la base de nombreuses applications dans ces domaines par le biais notamment des systèmes Intex et Unitex ou encore les outils de manipulations de modèles à états finis de Xerox [Karttunen *et al.*, 1997] et AT&T [Mohri, 1997]. Cependant ce système souffre d'inconvénients du fait de la structure du résultat produit qui est sous la forme d'un texte annoté. D'une part, la forme linéaire du texte produit en sortie ne permet pas de rendre compte des ambiguïtés d'analyses ainsi que des possibilités de chevauchement entre plusieurs analyses ; l'utilisation d'heuristiques (notamment la pondération des grammaires) permet, comme on l'a vu, de réduire les dommages dûs à cette limitation, mais seulement de façon partielle. D'autre part, le fait de produire un texte brut en sortie complique la mise en place de traitements nécessitant l'application de grammaires successives à un texte, puisqu'il est nécessaire à chaque étape, de réeffectuer les opérations coûteuses de pré-traitement (segmentation et étiquetage morpho-syntaxique) sur le texte nouveau obtenu

comme résultat de l'étape précédente.

Afin de pallier ces limitations, nous avons mis en place la fonctionnalité d'application d'un transducteur sur un automate du texte produisant en sortie un nouvel automate du texte enrichi par de nouveaux chemins produits par la grammaire. Pour ce type de traitement, les sorties présentes dans la grammaire ne peuvent pas être sous la forme de n'importe quelle chaîne de caractères, mais doivent être sous la forme d'une séquence de masques lexicaux. Ce sont ces symboles qui étiquettent les nouveaux chemins parallèles aux chemins reconnus par la grammaire.

Par exemple, les figures 3.11 et 3.12 présente une grammaire dans ce format pour la normalisation des textes. Cette grammaire rétablit les formes pleines des formes contractées et élidées retrouvées dans les textes.

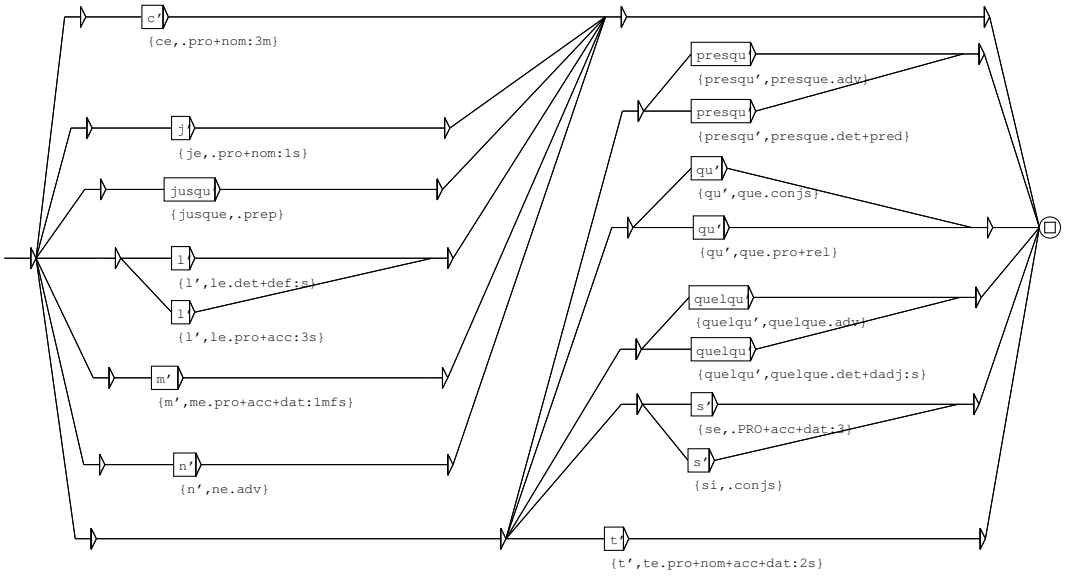

FIG. 3.11 – Normalisation des formes élidées

Notons que, pour ce type de transduction simple, les nouveaux chemins créés ne remplacent pas les anciens chemins qui ont permis leur reconnaissance mais sont placés dans l'automate en parallèle à ces derniers. Les chemins considérés comme superflus peuvent néanmoins être supprimés au moyen de grammaires ELAG ad hoc, ou encore en utilisant une variante plus puissante

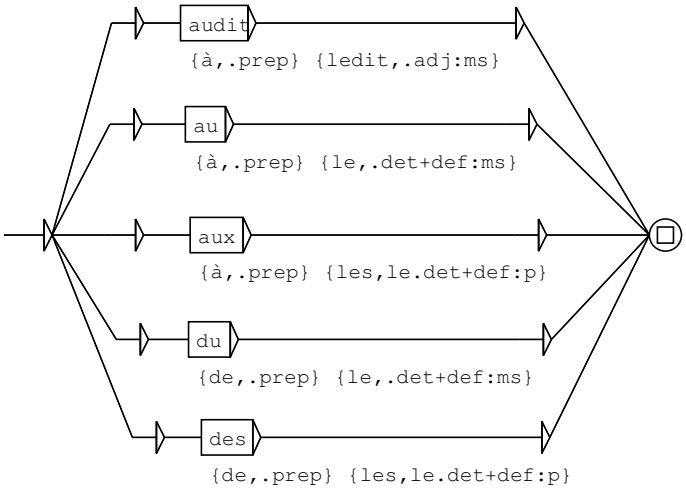

FIG. 3.12 – Normalisation des formes contractées

automate du texte normalisée

FIG. 3.13 – Résultat de l'application de la grammaire de normalisation

de la fonctionnalité d'application d'un transducteur sur un automate du texte que nous présentons dans la section qui suit.

3.6.5 Annotation de l'automate du texte

L'application d'une grammaire, telle que présentée dans la section précédente, est utile pour certains traitements simples (tels que la normalisation des textes), mais elle nécessite que les étiquettes des transitions créées lors de l'analyse soient écrites en une seule fois dans les sorties de la grammaire, alors qu'elles peuvent comporter plusieurs informations. Pour certains traitements (tels que la découpage en chunks, ou la reconnaissance d'entités nommées par exemple), qui consistent à reconnaître et étiqueter les occurrences des formes décrites dans la grammaire, il est nécessaire de distribuer l'étiquetage

des séquences reconnues sur l'ensemble du chemin parcouru lors de la reconnaissance du segment par la grammaire. Ce type de procédé peut être très fastidieux à écrire avec le formalisme des transducteurs tel que nous l'avons présenté précédemment puisqu'il oblige le grammairien à séparer entièrement les chemins parallèles de sa grammaire pour chaque combinaison de lexèmes qu'il juge pertinent d'étiqueter différemment. C'est pourquoi nous avons développé une variante plus puissante de la fonctionnalité d'application d'un transducteur sur l'automate du texte, qui donne plus de liberté sur la composition des étiquettes des transitions créées.

Pour ce type d'application, la forme des sorties des grammaires diffère de celle des grammaires utilisées pour la transduction plus classique : les étiquettes des chemins produits lors de l'application de la grammaire peuvent être décomposés sur plusieurs sorties en différents symboles qui spécifient les différents champs qui les composent.

Par exemple, la figure 3.14 présente une grammaire simple pour la reconnaissance et l'annotation des chunks nominaux dans l'automate du texte.

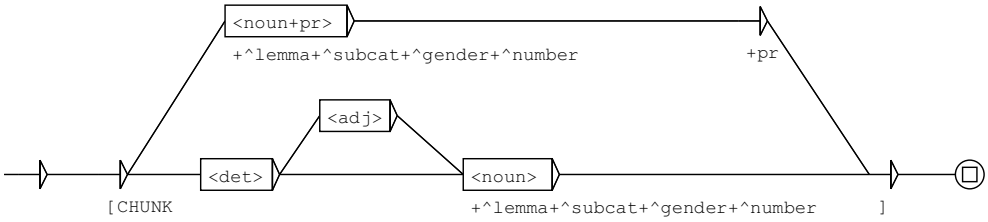

FIG. 3.14 – Grammaire simple pour la reconnaissance des CHUNK nominaux

Cette grammaire reconnaît des séquences telles que

Luc
les grandes idées
l'insatiable Marion

et les étiquettes en tant que chunk (étiquette CHUNK), en leur associant des traits spécifiant leur genre, leur nombre ou leur sous-catégorie sémantique :

{Luc,Luc.CHUNK+hum+pr+m+s}
{les grandes idées,idée.CHUNK+abst+f+p}
{l'insatiable Marion,Marion.CHUNK+hum+f+s}

Les sorties de la grammaire dans les chemins délimités par les balises [et] spécifient les différents traits qui composent l'étiquette syntaxique qui sera créée en sortie. Le premier symbole (CHUNK dans notre exemple), spécifie la catégorie syntaxique de la séquence (ce symbole doit être déclaré dans la description du jeu d'étiquettes) ; les symboles suivants spécifient les traits syntaxico-sémantiques. Ces derniers peuvent être de deux formes :

– soit ils sont composés de la valeur d'un trait syntaxico-sémantique précédé par un + (par exemple `+pr`). Dans ce cas, l'étiquette créée se verra attribuer cette valeur.
– soit ils sont composés du nom d'un attribut précédé par +^ (par exemple `+^gender`). Dans ce cas l'étiquette créée hérite de la valeur de cet attribut du lexème dans le texte qui a permis le franchissement de cette transition lors de l'analyse.

De plus, le fait de délimiter par des balises les limites des segments *étiquetés* par la grammaire permet de faire en sorte que ces limites ne coïncident pas nécessairement avec les limites des segments *reconnus* par la grammaire. Ceci permet de faire de l'étiquetage de formes en fonction de leur contexte d'apparition, voire même l'étiquetage de plusieurs segments discontinus lorsqu'ils apparaissent dans certains contextes spécifiés par la grammaire.

Par exemple, la figure 3.15 présente une grammaire simple utilisée pour l'identification des noms propres dans les textes. Celle-ci reconnaît *grosso modo* les séquences de mots commençant par une majuscule et les étiquette `N+pr`. Le fait de pouvoir dissocier les limites des segments reconnus par la grammaire et les limites des segments étiquetés par celle-ci, en combinaison avec l'utilisation de la pondération, nous a permis d'éviter d'étiqueter comme nom propre tous les mots se trouvant en début de phrase et qui commencent donc par une majuscule du fait des règles typographiques (sauf si ce mot est un mot inconnu). Cette heuristique simple est néanmoins très efficace, puisqu'elle nous a permis de réduire considérablement le bruit créé lors de la reconnaissance des noms propres, avec un taux de silence faible causé par

les cas pathologiques des noms propres homographes avec un mot commun et positionnés en début de phrase.

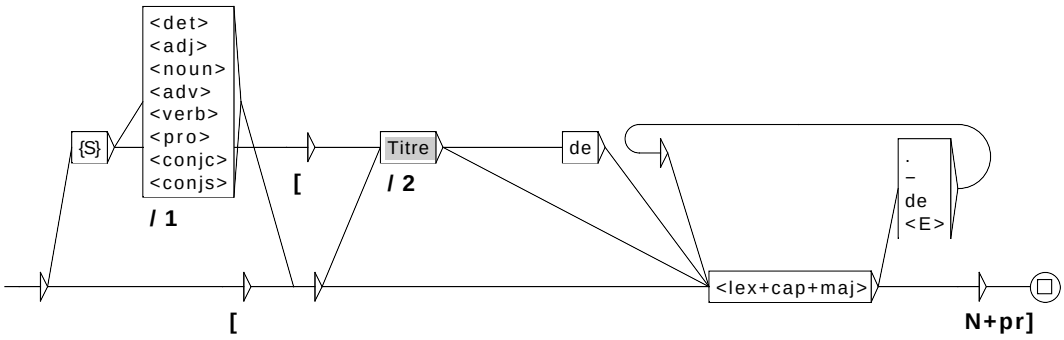

FIG. 3.15 – Grammaire pour la reconnaissance des noms propres

L'annotation de l'automate du texte par application d'une grammaire WRTN peut servir pour de nombreux traitements à différents niveaux d'analyse. Il peut par exemple être utilisé comme complément à l'étiquetage morpho-syntaxique pour la reconnaissance d'unités lexicales semi-figées dont les variations sont trop complexes pour être énumérées sous forme de liste mais qui peuvent être décrites dans des grammaires locales.

Ce procédé peut être itéré facilement pour procéder à la reconnaissance et l'étiquetage successifs de segments de plus en plus grands dans l'automate du texte.

Par exemple, la figure 3.16 présente l'automate de phrase précédent après l'application de la grammaire des adverbes de temps de M. Gross.

3.6.6 Extraction d'informations

Nous avons développé cette dernière application dans le cadre d'une démonstration des fonctionnalités de la plate-forme Outilex pour la conférence ACL 2006 [Blanc et Constant, 2006]. Cette application ne fait pas, à proprement

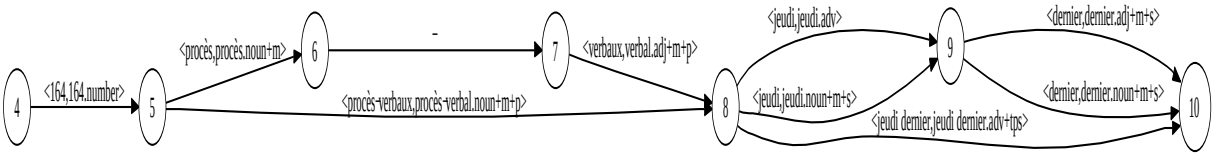

FIG. 3.16 – Résultat de l'application d'un transducteur sur l'automate du texte

parler, partie de la plate-forme, mais nous l'avons implémentée en réutilisant les fonctionnalités fournies par celle-ci de manière à démontrer les facilités d'extension d'Outilex dont l'architecture permet de réutiliser aisément certains de ces composants dans le cadre du développement d'applications TAL plus spécialisées.

Dans ce contexte, nous avons développé un module simple d'extraction d'informations par application de grammaires locales. Étant donné un type préétabli d'informations que l'on souhaite extraire des textes, nous partons du postulat que les différentes réalisations syntaxiques permettant de formuler des énoncés contenant ce type d'information n'admettent qu'un degré restreint de liberté. Il est ainsi possible de décrire toutes ces formes à l'aide de grammaires locales [Nakamura, 2005]. Les différentes entités qui participent à l'information extraite peuvent être identifiées à l'aide de balises incluses dans les sorties de la grammaire. Suivant ce principe, nous avons développé un module qui, étant donné le résultat de l'application d'une grammaire de ce type sur un texte, extrait de la forêt d'arbres d'analyse les informations reconnues par la grammaire et les formate dans des formulaires XML aux champs préétablis.

Afin de valider cette approche, nous avons, en collaboration avec Matthieu Constant, écrit une grammaire simple spécialisée dans l'extraction des informations biographiques (profession, éducation, date et lieu de naissance, etc.) dans les textes écrits en anglais. Les figures 3.17 et 3.18 présentent des extraits

de cette grammaire concernant les informations sur les postes qu'occupent les personnes citées dans les énoncés. En particulier, nous nous intéressons à identifier, lorsque ces informations sont disponibles, le nom de la personne, l'organisation pour laquelle elle travaille, la position qu'elle y occupe, ainsi que les dates de début et de fin pour cet emploi. Nous avons présenté un extrait des concordances obtenues par l'application de cette grammaire sur un corpus contenant une compilation d'articles du New York Times dans la figure 3.10 (section 3.6.2). L'utilisation de notre extracteur permet d'extraire automatiquement toutes les informations identifiées par la grammaire et de les formater dans des formulaires XML de la forme suivante :

```
<infos>
  <job>
    <who>Michael Mina</who>
    <position>executive chef</position>
    <organization>Aqua in San Francisco</organization>
    <date>since 1993</date>
    <segment>Michael Mina , the executive chef at Aqua in San Francisco
             since 1993</segment>
  </job>
</infos>
```

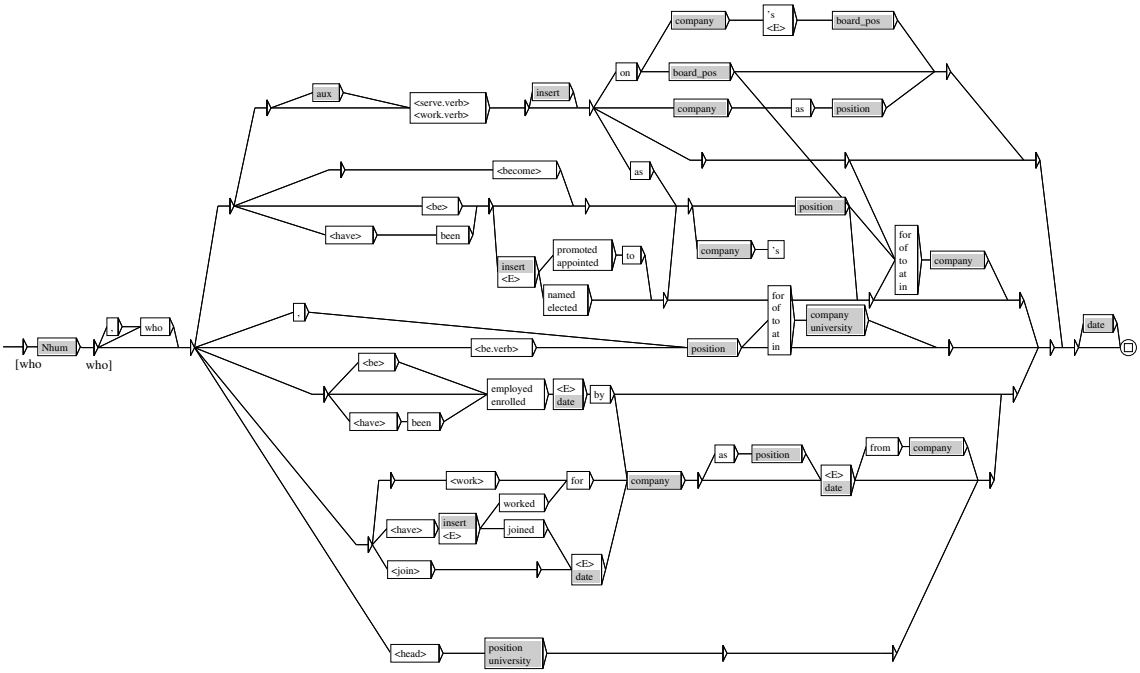

FIG. 3.17 – Extrait de la grammaire d'extraction d'information biographiques

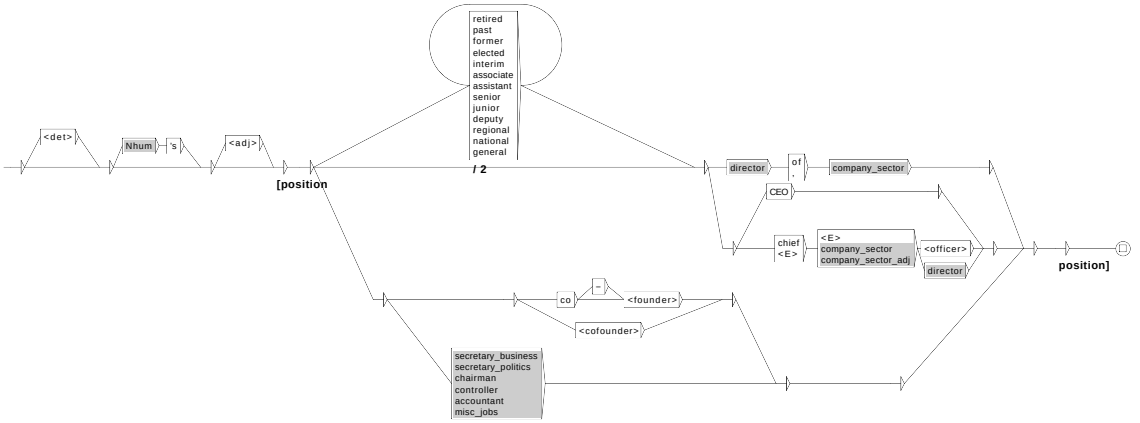

FIG. 3.18 – Sous-graphe `position`

Chapitre 4

Analyse syntaxique profonde

4.1 Introduction

Cette dernière partie présente quelques unes de nos expérimentations concernant *l'analyse syntaxique profonde* de textes en exploitant les propriétés syntaxiques des éléments prédicatifs du français encodées dans les tables du lexique-grammaire. A cet effet, nous avons développé une grammaire lexicalisé du français dans un formalisme à unification original (*RTN décorés*), qui est générée de manière semi-automatique à partir des tables du lexique-grammaire.

Nous présentons, dans un premier temps, la méthodologie du lexique-grammaire, en précisant en quoi les tables du lexique-grammaire sont une ressource précieuse pour le traitement automatique des langues. Nous présentons ensuite le formalisme grammatical des RTN décorés et nous le situons par rapport à d'autres formalismes utilisés couramment pour l'analyse syntaxique. Puis, nous présentons notre grammaire du français dans son état actuel et nous terminons en donnant quelques résultats préliminaires des évaluations de sa couverture lexicale et syntaxique.

4.2 Lexique-grammaire

Le lexique grammaire est une méthodologie pour l'étude empirique de la syntaxe des langues naturelles créée par Maurice Gross dont l'ouvrage fondateur est *Méthode en syntaxe* [Gross, 1975]. Cette méthodologie a pour cadre théorique la grammaire transformationnelle Harissienne [Harris, 1951, Harris, 1968] qui consiste en une approche mathématique de la linguistique, reposant sur des définitions rigoureuses et minimales. Dans ce cadre, le sujet d'étude est la phrase simple qui est considérée comme l'unité minimale de sens. Une phrase simple est composée d'un prédicat et de ses actants ou arguments, c'est-à-dire son sujet et ses compléments essentiels (par opposition aux compléments dits non-essentiels ou circonstanciels). Le prédicat est le noyau de la phrase, il s'agit le plus souvent d'un verbe plein. Ses arguments se distinguent par leur position dans la phrase (sujet, complément d'objet direct, complément prépositionnel, etc.) et par leur nature (groupe nominal humain, concret, phrase complétive, infinitive, etc.) :

Luc mange une pomme
Luc donne une pomme à Lea
(Que Lea soit partie+Partir en vacances) rejouit Max

Chaque prédicat sélectionne le nombre et la nature de ses arguments comme en attestent les phrases suivantes, qui sont grammaticalement incorrectes :

**Luc mange une pomme à Lea*
**Que Lea soit partie donne Max*
**Luc rejouit une pomme*

Le prédicat d'une phrase élémentaire peut être également un nom, un adjectif ou un adverbe. Dans ce cas, la phrase contient également un verbe considéré comme sémantiquement vide et appelé *verbe support* dans ce contexte :

Luc prend la *douche* (= la douche que Luc prend)
Lea est *fière* de son fils
Cet évenement s'est déroulé *(tranquillement+jeudi dernier)*

Certaines expressions dites figées peuvent également jouer le rôle de prédicat d'une phrase. Elles ne présentent en surface pas de différences avec les constructions dites libres, mais certains de leurs éléments sont contraints et n'admettent pas ou peu de variations lexicales :

Luc prend la tangente (= * la tangente que Luc prend)
Luc prend le taureau par les cornes
Lea fait face à ce problème

Une phrase simple peut être sujette à une transformation (telle que la mise au passif, ou la pronominalisation, l'effacement ou le deplacement d'un argument par exemple) produisant ainsi une nouvelle phrase considérée comme transformationnellement équivalente :

Luc a rendu la télé à Lea
= *Luc a rendu à Lea la télé [permutation]*
= *La télé a été rendue à Lea par Luc [passif]*
= *La télé a été rendue à Lea [effacement]*
= *La télé lui a été rendue [pronominalisation]*

D'autre type de transformations, dites transformations binaires, permettent de construire des phrases complexes à partir de deux phrases simples (ou complexes). Il s'agit par exemple

– de la coordination :

Luc danse ; Marie s'ennuie
= *Luc danse et Marie s'ennuie*

– de la subordination :

Luc danse tandis que Marie s'ennuie

– de la relativation :

Luc aime la danse ; Lea exècre Luc
= *Luc, que Lea exècre, aime la danse*

Selon Harris, les phrases simples sont des unités élémentaires qui permettent de construire tous types d'énoncés complexes par applications successives

de transformations. Inversement, tout énoncé en langue naturelle peut être décomposé en un ensemble de phrases élémentaires qui sont les unités de sens qui le composent.

Maurice Gross a mis en avant l'importance du lexique par rapport à la grammaire. En effet, comme même Chomsky l'observait [Chomsky, 1965], les transformations syntaxiques, même les plus générales, sont sujettes à de fortes contraintes au niveau lexical. Ainsi, une description complète de la syntaxe d'une langue naturelle ne consiste pas en un ensemble de règles syntaxiques générales mais nécessite également, et de manière aussi importante, une description détaillée pour chaque élément du lexique des formes et des transformations syntaxiques qu'il accepte ou n'accepte pas. Ainsi, Maurice Gross a entrepris, avec son équipe au LADL, une description des propriétés syntaxiques des éléments prédicatifs du français : verbes, noms, adjectifs, adverbes et phrases figées. Pour chaque prédicat, ont été étudiées de manière sytématique ses propriétés de sous-catégorisation (le nombre et la nature de ses arguments) ainsi que ses propriétés transformationnelles. Toutes ces descriptions ont été encodées dans des dictionnaires syntaxiques sous la forme de tables, dites tables du lexique grammaire. Chaque table regroupe un ensemble de prédicats en fonction de propriétés définitionnelles, correspondant souvent à la structure de la phrase canonique. La figure 4.1, par exemple, contient un extrait de la table 9 qui regroupe l'ensemble des verbes qui rentrent dans la construction *N0 V que P à N1 (= : Luc dit qu'il va bien à Lea)*. Chaque ligne de la table correspond à un prédicat ; chaque colonne correspond à une propriété. Une valeur booléenne (indiquée par un + ou −) à chaque intersection d'une ligne et d'une colonne indique si telle entrée accepte ou non telle propriété. Les chercheurs du lexique-grammaire ont ainsi codé 12 000 emplois de verbes [Gross, 1975, Boons *et al.*, 1976b, Boons *et al.*, 1976a, Guillet et Leclère, 1992], 10 000 emplois de noms prédicatifs [Giry-Schneider, 1978][Labelle, 1974][Meunier, 1981][Vivès, 1983] [de Negroni-Peyre, 1978 [Gross, 1989]. Les tables des adjectifs sont en cours de construction et il existe également des tables de phrases figées (M. Gross, 1984) comprenant une vingtaine de milliers d'entrées.

Le but de notre travail consiste à exploiter les informations de sous-catégorisation et les propriétés transformationnelles présentes dans les tables du lexique-grammaire afin d'identifier les phrases simples (c'est-à-dire les prédicats syntaxiques et leurs arguments) dans les textes français. Par exemple,

N0 = : Nhum	N0 = : Nnr	N0 = : le fait que P	N0 = : V1 W	N0 = : V2 W	Aux = : avoir	entry	N0 V	N0 V W	N0 pousser Dét V-m	N0 pousser Dét V-E	N0 V contre Nhum	N0 V après Nhum	N0 V N1	N1 = : que P	N1 = : V0 W	N1 = : Aux V0 W	N1 = : de V0 W	(N1) (être Adj)	(être Adj) (que P)	que N0 V (être Adj)	(N1) (Adj)	(Adj) (que P)	que N0 V (Adj)	Nég interr = > subj	Impératif = > subj	N1 = : V0 W	N1 = : V2 W
+	-	-	-	-	+	développer	-	-	-	-	-	-	+	+	-	+	-	-	+	+	-	-	-	+	-	-	-
+	-	-	-	-	+	dévider	-	-	-	-	-	-	+	+	-	+	-	-	+	+	-	-	-	+	-	-	-
+	+	+	+	+	+	dévoiler	-	-	-	-	-	-	+	+	-	+	-	-	+	+	-	-	-	-	-	-	-
+	+	+	-	+	+	dicter	-	+	-	-	-	-	+	+	-	+	-	-	+	+	-	-	-	+	-	+	-
+	-	-	-	-	+	dicter	-	-	-	-	-	-	+	+	-	+	-	-	-	-	-	-	-	-	-	+	-
+	+	+	-	+	+	diffracter	-	-	-	-	-	-	+	+	-	+	-	-	+	+	-	-	-	+	-	+	-
+	-	-	-	-	+	diffuser	-	-	-	-	-	-	+	+	-	+	-	-	-	-	-	-	-	-	-	-	-
+	+	+	-	+	+	dire	-	-	-	-	-	-	+	+	+	+	-	-	+	+	-	-	-	+	-	+	-
+	+	+	-	+	+	disputer	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	+	-
+	+	+	+	+	+	dissimuler	-	-	-	-	-	-	+	+	-	+	-	-	-	-	-	-	-	+	-	-	-
+	+	+	-	+	+	divulguer	-	-	-	-	-	-	+	+	-	+	-	-	+	+	-	-	-	-	-	-	-
+	-	-	-	-	+	donner	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
+	+	+	-	+	+	économiser	-	+	-	-	-	-	+	-	-	-	-	-	-	-	-	-	-	-	-	+	-
+	-	-	-	-	+	écrire	-	+	-	-	+	-	-	+	-	+	-	-	+	+	-	-	-	+	-	+	-
+	-	-	-	-	+	édicter	-	-	-	-	-	-	+	+	-	-	-	-	-	-	-	-	-	-	-	+	-
+	-	-	-	-	+	émettre	-	-	-	-	-	-	+	+	-	+	-	-	+	+	+	-	+	+	-	-	-
+	+	+	-	+	+	enjoindre	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	+	-
+	+	+	-	+	+	enlever	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	+	-
+	-	-	-	-	+	énoncer	-	-	-	-	-	-	+	+	-	+	-	-	+	+	-	-	-	+	-	+	-

FIG. 4.1 – Extrait de la table 9

étant donné le texte suivant (extrait de [Goscinny et Sempé, 2006]) :

> Ce matin, Geoffroy est entré dans la cour de l'école, il s'est arrêté et il nous a crié : «Eh ! Les Gars ! Venez voir ce que j'ai !» C'est drôle, chaque fois que Geoffroy amène quelque chose à l'école, il s'arrête à l'entrée de la cour de récré, il nous crie : « Eh ! Les Gars ! Venez voir ce que j'ai ! » Alors, nous y sommes allés et il avait un stylo.

Nous souhaitons en extraire automatiquement les phrases élementaires qui le composent (afin de simplifier notre exemple, nous considérons la séquence de phrases au discours direct comme une seule unité, et nous omettons de

reporter les temps de conjugaison) :

P' : *«Eh ! Les Gars ! Venez voir ce que j'ai ! »*
P1 : *Geoffroy entre dans la cour de l'école*
P2 : *Geoffroy s'arrête*
P3 : *Geoffroy nous crie P'*
P4 : *P1, P2 et P3 se passe ce matin*
P5 : *Geoffroy amène quelque chose à l'école*
P6 : *P1, P2 et P3 se passe chaque fois que P5*
P7 : *P6 est drôle*
P8 : *nous allons à l'entrée de la cour de récré*
P9 : *Geoffroy a un stylo*

Nous pouvons également représenter ces phrases simples sous la forme de prédicats logiques :

P1 : `entrer(Geoffroy, dans(cours de l'école))`
P2 : `se-arrêter(Geoffroy)`
P3 : `crier(Geoffroy, P')`
P4 : `ce-matin(et(P1,P2,P3))`
P5 : `amener(Geoffroy, quelque chose, école)`
P6 : `chaque-fois(et(P1, P2, P3), P5)`
P7 : `drôle(P6)`
P8 : `aller(nous, entrée de la cour de récré)`
P9 : `avoir(Geoffroy, stylo)`

Nous n'avons pas résolu tous les problèmes nous permettant de réaliser une telle analyse automatiquement, cependant, les travaux présentés dans ce chapitre nous ont permis d'avancer sérieusement dans ce sens.

L'analyse syntaxique ne constitue pas une fin en soi, mais est souvent une étape nécessaire pour le développement d'applications de plus haut niveau en traitement des langues, tels que l'analyse sémantique, les systèmes de réponses à des questions, la génération de résumés de texte ou la traduction automatique. Toutes ces applications pourraient bénéficier d'un module d'analyse syntaxique exploitant les descriptions linguistiques fines et à large

couverture telles que celles qui ont été élaborées dans le cadre du lexique-grammaire [Gardent *et al.*, 2005].

4.3 Formalisme Grammatical

4.3.1 Stuctures de traits et unification

Une structure de traits est constituée d'un ensemble de traits sous la forme de couples attribut-valeur. La valeur d'un trait peut être de deux formes. Il s'agit soit d'une valeur atomique (dans ce cas elle prend la forme d'un symbole ou plus généralement d'une disjonction de symboles atomiques), soit d'une structure de traits enchâssée. Traditionnellement, on représente une structure de traits par des matrices entre crochets. La figure 4.2 présente deux exemples de structures de traits. La première est une structure de traits simple dans le sens où chaque trait à une valeur atomique. Le trait *cat* par exemple a pour valeur le symbole *det* et la valeur de l'attribut *genre* est constituée de la disjonction des deux symboles atomiques *masc* et *fem*. La matrice à droite, quant à elle, représente une structure de traits complexe puisque la valeur du trait *accord* est elle-même une structure de traits enchassée.

$$\begin{bmatrix} cat : det \\ genre : masc|fem \\ nombre : pl \end{bmatrix} \quad \begin{bmatrix} cat : det \\ accord : \begin{bmatrix} genre : masc \\ nombre : plur \end{bmatrix} \end{bmatrix}$$

FIG. 4.2 – Exemples de structures de traits

Pour faciliter la formalisation de certains phénomènes de syntaxe, nous manipulerons des structures de traits dont la valeur d'un trait peut être également un ensemble de valeurs (atomiques ou complexes). Cette extension ne modifie pas notre définition des structures de traits puisque nous représentons ce type de valeur par des structures de traits enchassées à deux attributs *head* et *tail*, à la manière d'une liste chaînée (cf. figure 4.3). Pour des raisons de lisibilité, nous affichons ce type de valeur sous la forme d'une liste encadrée par les symbole $<$ et $>$:

$$\begin{bmatrix} head : Pierre \\ tail : \begin{bmatrix} head : Paul \\ tail : \begin{bmatrix} head : Jacques \\ tail : [] \end{bmatrix} \end{bmatrix} \end{bmatrix} \quad \text{<Pierre,Paul,Jacques>}$$

FIG. 4.3 – Représentations des traits à valeur ensembliste

L'ordre dans lequel apparaissent les attributs dans une structure de trait n'est pas important. Par exemple, les deux structures de traits suivantes sont strictement équivalentes.

$$\begin{bmatrix} cat : det \\ accord : \begin{bmatrix} genre : masc \\ nombre : plur \end{bmatrix} \end{bmatrix} \quad \begin{bmatrix} accord : \begin{bmatrix} nombre : plur \\ genre : masc \end{bmatrix} \\ cat : det \end{bmatrix}$$

FIG. 4.4 – Structure de traits équivalentes

Une structure de traits ne peut pas avoir deux traits distincts pour un même attribut. Par exemple, la matrice à gauche dans la figure 4.5 ne représente pas une structure de traits bien formée ; la structure de droite est bien formée puisque les deux attributs *genre* n'apparaissent pas au même niveau d'enchâssement dans la structure.

$$\begin{bmatrix} genre : masc \\ genre : fem \end{bmatrix} \quad \begin{bmatrix} genre : masc \\ accord : [\ genre : fem\ ] \end{bmatrix}$$

FIG. 4.5 – Exemple de structures de traits mal-formées et bien formées

Enfin, plusieurs traits peuvent partager une même valeur dans une structure de traits. Dans ce cas la structure de traits est dite *réentrante*. Par exemple dans la structure de trait A de la figure 4.6, les deux attributs *accord* sont coïndicés par l'indice 1 et partagent donc le même trait $[\ num : sing\ ]$.

$$A$$
$$\left[\begin{array}{l} Det : \left[\ accord : (1) \left[\ num : sing\ \right]\ \right] \\ N : \left[\ accord : (1)\ \right] \end{array}\right]$$

FIG. 4.6 – Structure de traits A réentrante

Cette structure est différente de la structure B, dans laquelle les valeurs de ses deux attributs *accord* sont identiques mais ne sont pas partagées.

$$B$$
$$\left[\begin{array}{l} Det : \left[\ accord : \left[\ num : sing\ \right]\ \right] \\ N : \left[\ accord : \left[\ num : sing\ \right]\ \right] \end{array}\right]$$

FIG. 4.7 – Structure de trait B non réentrante

Subsomption

La relation de *subsomption* définit une relation d'ordre partiel sur l'ensemble des structures de traits.

On dit qu'une structure de traits A subsume une structure de traits B (ce qui est noté $A \sqsubseteq B$) si et seulement si :

- tous les traits à valeur atomique présents dans A sont présents dans B avec la même valeur ou une valeur plus précise.
- pour tout trait à valeur non atomique dans A, ce trait est présent dans B et la valeur de ce trait dans A subsume la valeur de ce même trait dans B ;
- si deux traits ont une valeur partagée dans A, alors ces mêmes traits ont une valeur partagée dans B.

Informellement, on dit que A subsume B si toutes les informations contenues dans A sont également contenues dans B ou encore si B est plus (ou également) précise que A.

Par exemple, étant données les structures de traits suivantes (figure 4.8) :

$$\underset{F_1}{\left[\ genre : masc|fem\ \right]} \qquad \underset{F_2}{\left[\begin{array}{l} num : pl \\ genre : masc|fem \end{array}\right]} \qquad \underset{F_3}{\left[\begin{array}{l} num : pl \\ genre : masc \end{array}\right]}$$

FIG. 4.8 – Exemple de subsomptions

La structure F_1 subsume F_2 et F_3 mais F_2 et F_3 ne subsument pas F_1 puisqu'elles contiennent un attribut *genre* qui est absent dans F_1. De même F_2 subsume F_3 mais F_3 ne subsume F_2 puisque la valeur de l'attribut *genre* est plus précise dans F_3 que dans F_2.

La relation de subsomption définit une relation d'ordre partielle sur l'ensemble des structures de traits, dans le sens où elle ne permet pas de mettre en relation tous couples de structures de traits. En effet, étant données les structures de traits suivantes :

$$\left[\begin{array}{l} num : sg \\ genre : masc \end{array}\right] \qquad \left[\begin{array}{l} person : 3 \\ num : sg \end{array}\right]$$

FIG. 4.9 – Structures de traits qui ne se subsument pas

aucune des structures ne subsume l'autre, puisque chacune contient une information absente dans l'autre structure.

Enfin, étant données les structure A et B vues précédemment (figures 4.6 et 4.7), la structure B subsume la structure A mais A ne subsume pas B. En effet, toutes les informations présentes dans B sont bien présentes dans A, en revanche, les deux attributs *accord* enchassés dans les attributs DET et N de A partagent la même valeur, ce qui n'est pas le cas dans la structure B.

Unification

L'unification est une opération fondamentale sur les structures de traits ; elle permet de combiner les informations présentes dans deux structures de traits lorsqu'elles sont compatibles.

Formellement, étant données deux structures de traits A et B, le résultat de l'unification de A et B (notée $A \sqcup B$) est la structure de traits minimale F telle que A et B subsument F. Si une telle structure n'existe pas, l'unification échoue et son résultat est notée $\bot$.

Informellement, l'unification de deux structures de traits produit la plus petite structure de traits qui contient toutes les informations contenues dans les deux structures, si ces informations sont compatibles.

Par exemple, étant donné les structures F_1 et F_2 suivantes

$$F_1 \qquad\qquad F_2$$
$$\begin{bmatrix} cat : N \\ accord : \begin{bmatrix} num : sg \end{bmatrix} \end{bmatrix} \qquad \begin{bmatrix} cat : N \\ accord : \begin{bmatrix} genre : fem \end{bmatrix} \end{bmatrix}$$

FIG. 4.10 – Structures F_1 et F_2

le résultat de l'unification de ces deux structures est la structure suivante :

$$F_1 \sqcup F_2$$
$$\begin{bmatrix} cat : N \\ accord : \begin{bmatrix} num : sg \\ genre : fem \end{bmatrix} \end{bmatrix}$$

FIG. 4.11 – $F_1 \sqcup F_2$

En revanche, étant données les deux structures suivantes :

$$\begin{array}{cc} F_3 & F_4 \\ \left[\ cat : N\ \right] & \left[\ cat : Det\ \right] \end{array}$$

FIG. 4.12 – Structure de traits incompatibles

l'unification de ces deux structures échoue, car il n'existe aucune structure de traits qui contienne les informations contenues dans F_3 et F_4, car celles-ci sont contradictoires. $F_3 \sqcup F_4 = \bot$.

Enfin, si nous reprenons les structures A réentrante et B non réentrante des figures 4.6 et 4.7 et que l'on unifie ces deux structures avec la structure C ci-dessous, les résultats seront bien différents :

$$C$$
$$\left[\ Det : \left[\ accord : \left[\ genre : masc\ \right]\ \right]\ \right]$$
$$A \sqcup C :$$
$$\left[\begin{array}{l} Det : \left[\ accord : 1 \left[\begin{array}{l} num : sing \\ genre : masc \end{array}\right]\ \right] \\ N : \left[\ accord : (1)\ \right] \end{array}\right]$$
$$B \sqcup C :$$
$$\left[\begin{array}{l} Det : \left[\ accord : \left[\begin{array}{l} num : sing \\ genre : masc \end{array}\right]\ \right] \\ N : \left[\ accord : \left[\ num : sing\ \right]\ \right] \end{array}\right]$$

FIG. 4.13 – Résultats de l'unification

Ainsi, dans la structure obtenue par unification de A et B, le trait *genre* enchassé sous l'attribut *accord* de l'attribut N est spécifié (et partage sa valeur avec le même attribut enchassé sous l'attribut *Det*), ce qui n'est pas le cas pour la structure obtenue après unification de B et C.

4.4 Les RTN décorés comme formalisme grammatical

Le formalisme des RTN décorés (DRTN) est une extension du formalisme WRTN dans lequel nous avons intégré une composante d'unification. Une grammaire DRTN est une grammaire WRTN dont les sorties sont décorées par des équations sur les traits. Ces équations permettent de construire les structures de traits associées à chaque constituant syntaxique lors de l'analyse. Comme nous le verrons, l'utilisation des structures de traits nous a permis de formaliser et résoudre différents phénomènes de syntaxe, des plus simples comme les contraintes d'accord grammatical jusqu'à des phénomènes plus complexes tels que la résolution de certaines coréférences ou les dépendances à distance.

Le résultat de l'analyse d'une phrase par une grammaire DRTN est constitué d'une forêt d'arbres syntaxiques qui représentent les possibles découpages de la phrase en constituants syntaxiques ; à chaque arbre est associée une structure de traits dans laquelle sont stockées les relations grammaticales entre ces constituants qui ont été calculées durant l'analyse à l'aide des équations sur les traits qui décorent les transitions de la grammaire. La représentation du résultat de l'analyse par une structure de traits nous permet plus d'expressivité qu'avec un simple arbre syntaxique. En particulier, nous nous en servons, dans le cadre de ce travail, pour identifier dans les textes les actants sémantiques de chaque prédicat indépendamment de leur position syntaxique dans la phrase.

Par exemple, la figure 4.17 et les figures suivantes (page 127) présentent une grammaire simplifiée pour le verbe *donner*. Le format et la sémantique des équations sur les traits présentes dans les sorties de la grammaire seront expliqués plus en détail dans la suite.

Cette grammaire analyse par exemple, les deux phrases suivantes :

Luc a escroqué une forte somme à Lea
Luc a escroqué Lea d'une forte somme

Ces deux constructions sont considérées, dans le cadre du lexique-grammaire, comme transformationnellement équivalentes ; la seconde étant obtenue comme le résultat de l'application d'une transformation *de croisement des arguments* sur la première. Ainsi, les actants *Luc*, *Lea* et *somme* jouent le même rôle argumental par rapport au prédicat *escroquer* dans les deux phrases ; cette équivalence n'apparaît pas au niveau des arbres syntaxiques obtenus à l'issue de l'analyse de ces deux phrases (figures 4.14 et 4.15) :

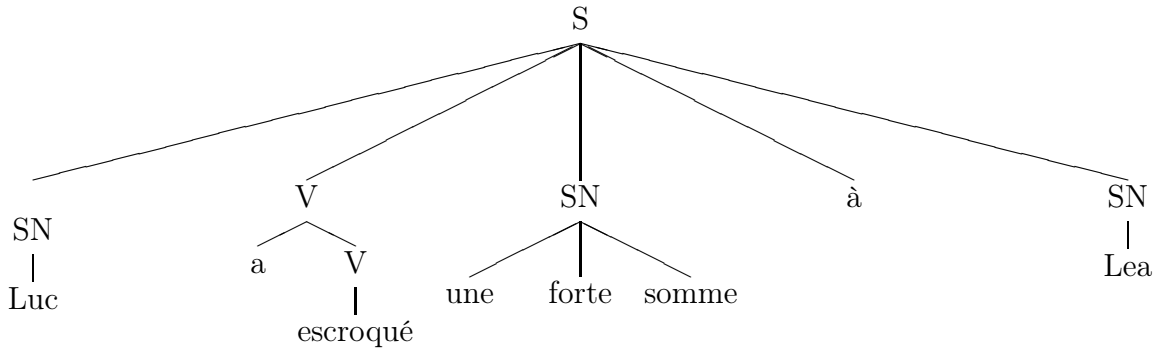

FIG. 4.14 – Arbre syntaxique associé à la phrase *Luc a escroqué une forte somme à Lea*

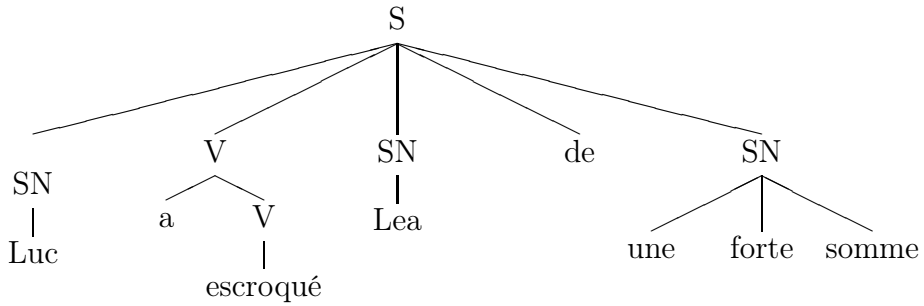

FIG. 4.15 – Arbre syntaxique associé à la phrase *Luc a escroqué Lea d'une forte somme*

En revanche, à l'issue de l'analyse, nous associons à chacun de ces arbres de dérivation une structure de traits dans laquelle ces relations d'équivalence

entre le prédicat et ses arguments sont bien explicitées, puisque nous obtenons exactement la même structure de traits comme résultat de l'analyse des deux phrases :

$$\begin{bmatrix} Pred : (1)escroquer \\ V : \begin{bmatrix} lemma : (1) \\ mode : ind \end{bmatrix} \\ n0 : Luc \\ n1 : somme \\ n2 : Lea \end{bmatrix}$$

FIG. 4.16 – Structure de traits obtenue comme résultat de l'analyse des deux phrases transformationnellement équivalentes

C'est en ce sens que nous parlons de syntaxe *profonde*.

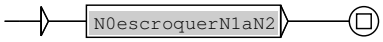

FIG. 4.17 – Graphe non terminal P (axiome de la grammaire)

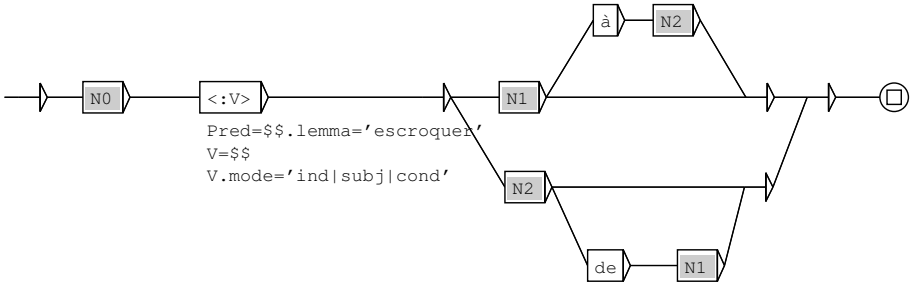

FIG. 4.18 – Sous-graphe de structuration N0escroquerN1aN2

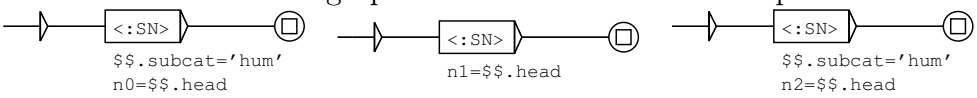

FIG. 4.19 – Sous-graphes N0, N1, N2

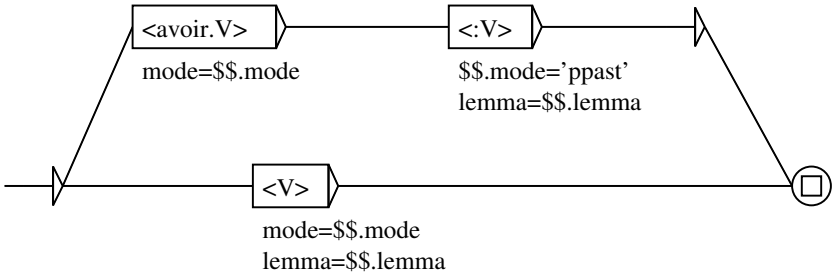

FIG. 4.20 – Non terminal V

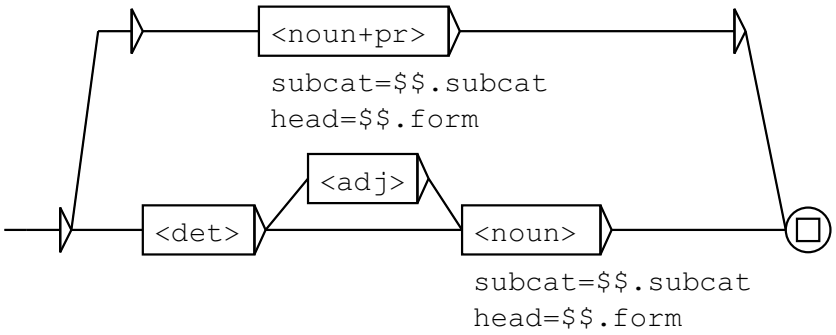

FIG. 4.21 – Non terminal SN

Les différentes conventions d'appel à un sous-graphe

Nous faisons la distinction dans notre formalisme entre *sous-graphe de structuration* et *sous-graphe non terminal*. Ces deux types de graphes se distinguent par la forme avec laquelle ils sont instanciés depuis leur graphe parent (voir par exemple notre grammaire jouet page 127) :

- la convention classique d'appel à un sous-graphe (notée :X) qui apparait en grisé dans les figures ; ce type d'appel à un sous-graphe ne correspond pas à la reconnaissance d'un non-terminal de la grammaire. En effet, lors de la compilation de celle-ci, les sous-graphes appelés avec ce type de convention sont directement intégrés dans leur graphe parent à l'aide des méthodes d'aplatissement de grammaire que nous avons décrites dans la section 3.4.
- l'appel à un non-terminal noté < :X> ; dans ce cas X est un symbole non terminal dont les formes sont décrites dans un automate de la grammaire.

Les sous-graphes non terminaux servent aux appels (éventuellement récursifs) du RTN et correspondent à des noeuds de l'arbre de dérivation. Les sous-graphes de structuration servent à assurer la compacité et la lisibilité des éléments de la grammaire et ne peuvent pas s'appeler récursivement entre eux (sauf par l'intermédiaire d'un sous-graphe non terminal).

On peut voir que le graphe `N0escroquerN1aN2` (figure 4.18) utilise les deux types de conventions d'appel : les 3 arguments du verbe sont reconnus dans des sous-graphes appelés par la convention classique :N0, :N1, :N2 ; alors que le noyau verbal de la phrase est reconnu par le constituant syntaxique V de la grammaire décrit dans la figure 4.20.

La figure 4.22 présente l'arbre syntaxique obtenu après l'analyse de la phrase : *Cet homme a escroqué une fleur à Lea.*

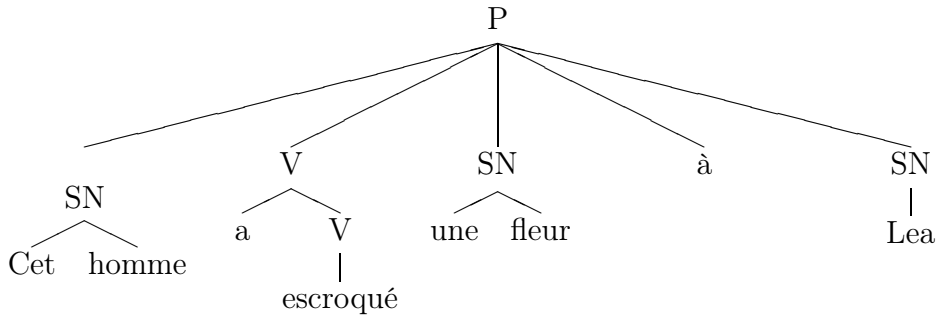

FIG. 4.22 – Arbre de dérivation

On constate que les non terminaux de la grammaire apparaissent comme les noeuds internes de l'arbre de dérivation, ce qui n'est pas le cas des sous-graphes `N0escroquerN1aN2`, `N0`, `N1` et `N2` qui eux n'apparaissent pas.

Par ailleurs, si l'on souhaite enrichir notre grammaire de nouvelles constructions, les sous-graphes décrivant les actants du verbe peuvent directement être réutilisés, comme par exemple dans le graphe escroquer-passif de la figure 4.23 qui décrit certaines constructions à la voix passive pour le verbe *escroquer*.

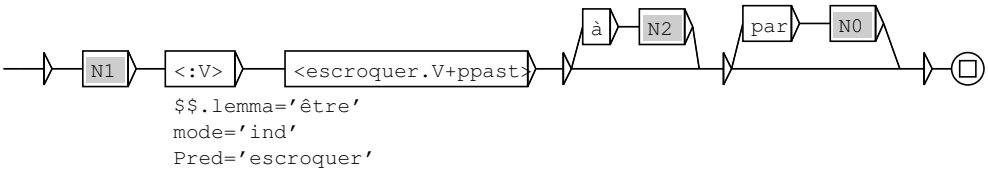

FIG. 4.23 – Constructions passives

Nous modifions également le graphe principal P, axiome de la grammaire, de manière à ce qu'il prenne en compte ces nouvelles descriptions (cf. figure 4.24). Notre grammaire analyse maintenant de nouvelles constructions telles que :

> *une (belle+sacrée) somme a été escroquée à l'état (par le premier ministre+E).*

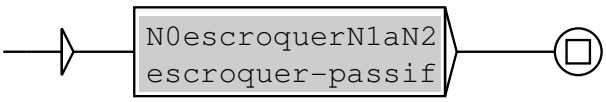

FIG. 4.24 – Graphe principal modifié

Ainsi, grâce à ce système à deux conventions d'appel à des sous-graphes, nous obtenons une indépendance entre la structuration de la grammaire en graphes et la structure des arbres syntaxiques. Ceci permet à l'auteur de la grammaire de factoriser certaines descriptions linguistiques dans des graphes en vue de leur réutilisation dans des descriptions de tailles plus importantes sans que ces choix influencent directement la forme des structures produites lors de l'analyse.

Equations sur les traits

Les équations sur les traits (ou *équations fonctionnelles*) spécifient des contraintes sur les structures de traits qui sont associées à chaque constituant syntaxique reconnu par la grammaire. Ces contraintes permettent de construire ces structures de traits durant l'analyse ou éventuellement de faire échouer l'analyse dans le cas où des contraintes ne peuvent pas être satisfaites.

Les équations utilisent la notion de *chemin* dans une structure de traits (ou *chemin fonctionnel*) pour identifier un trait dans une structure de traits. Un chemin fonctionnel consiste en la séquence des attributs (séparés par un '.') qu'il faut franchir en partant de la racine de la structure de traits pour parvenir à un trait particulier. Par exemple, étant donnée la structure suivante :

$$\left[\begin{array}{l} det : (1)\left[\ accord : \left[\begin{array}{l} genre : masc \\ num : pl \end{array}\right]\ \right] \\ n : (1) \end{array}\right]$$

Le chemin *det.accord.genre* identifie la valeur du trait d'attribut *num* enchassé dans l'attribut *accord* enchassé dans l'attribut *det*. Ce même trait est d'ailleurs identifié par le chemin *n.accord.genre* puisque la structure est réentrante et que ces deux traits partagent la même valeur.

Par défaut, un chemin dans une équation fonctionnelle référence un trait dans la structure de traits associée au non terminal décrit par le graphe courant. Cependant, le symbole $$ permet de référencer la structure de trait associée au symbole contenu dans la boîte sous laquelle l'équation apparaît. Par exemple dans le graphe de reconnaissances des SN de la figure 4.21, l'équation *subcat=$$.subcat* sous la boîte étiquetée par <noun> permet d'unifier la valeur de l'attribut *subcat* de la structure de traits associée au non terminal courant SN (chemin : *subcat*) avec la valeur de l'attribut *subcat* de la structure associée au nom reconnu par le masque lexical <noun> (chemin : *$$.subcat*).

Dans le cas où le symbole qui étiquette la transition est un symbole non terminal (de la forme < :X>), la structure de traits référencée par le symbole $$ est la structure associée au non terminal X reconnu lors du franchissement de la transition ; cette structure est construite durant l'analyse de ce constituant.

Dans le cas où le symbole qui étiquette la transition est un symbole terminal (c'est-à-dire un masque lexical), le symbole $$ référence la structure de traits qui est construite à partir du mot étiqueté du texte (représenté par un masque lexical) qui a permis le franchissement de cette transition. Un masque lexical étant lui-même représenté par une structure composée de champs sous la forme de couples attribut-valeur, la transformation d'un masque en une structure de traits est triviale : l'opération consiste à reformater ces informations sous la forme d'une structure de traits simple dont les traits ont une valeur atomique, les noms des attributs étant décrits dans la description du jeu d'étiquettes. Par exemple, le mot étiqueté <corps,corps.noun+conc+m+s+p> se traduit sous la forme de la structure suivante :

$$\begin{bmatrix} form : corps \\ lemma : corps \\ CAT : noun \\ subcat : conc \\ gender : m \\ number : s|p \end{bmatrix}$$

FIG. 4.25 – Structure de traits associée au nom *corps*

L'utilisation du symbole $$ dans une équation fonctionnelle n'a pas de sens si elle est présente dans les sorties d'une boîte qui contient un appel à un sous-graphe de structuration (de la forme :X). En effet, comme on l'a vu précédemment, un graphe appelé par cette convention est directement intégré dans son graphe parent et par conséquent ne décrit pas un constituant syntaxique de la grammaire. A ce titre, il n'a pas de structure associée propre et les équations fonctionnelles qu'il contient contribuent directement à la construction de la structure de traits associée au constituant syntaxique décrit dans le graphe à partir duquel il a été intégré.

Les équations sur les traits qui décorent les grammaires DRTN peuvent être de plusieurs formes :

contrainte d'unification

Les contraintes d'unification sont la forme la plus courante des équations fonctionnelles. C'est d'ailleurs la seule forme que nous avons utilisée dans nos exemples jusqu'à présent. Une contrainte d'unification a une des deux formes suivantes :

– $chemin_1 = chemin_2$
– $chemin_1 =$ *'valeur'*

où $chemin_1$ et $chemin_2$ sont des chemins fonctionnels et *valeur* est un symbole atomique ou une disjonction de symboles atomiques. Une contrainte de ce type permet d'unifier la valeur de deux traits spécifiés par des chemins

fonctionnels (première forme), ou d'unifier la valeur d'un trait avec une valeur atomique précisée dans la grammaire (seconde forme). Dans le cas où l'unification échoue parce que les valeurs sont incompatibles, l'analyse échoue.

Par exemple, l'équation *$$.subcat='hum'* sous la boîte étiquetée < :SN> dans le graphe N0 (figure 4.19) permet de contraindre la reconnaissance aux constituants de catégorie SN ayant un attribut *subcat* compatible avec la valeur *hum* ; cet attribut étant lui-même hérité du mot de tête du SN par l'équation *subcat=$$.subcat* (figure 4.21).

De même, nous pouvons améliorer le graphe SN de manière à vérifier l'accord en genre et en nombre entre le déterminant, l'éventuel adjectif et le nom de tête et faire remonter ces informations au niveau de la structure de traits associée à ce constituant. Le graphe devient alors celui de la figure 4.26.

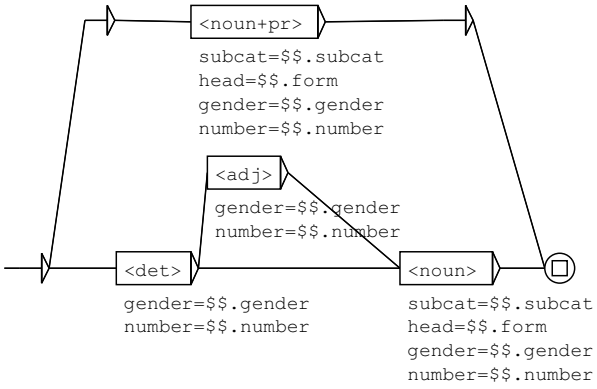

FIG. 4.26 – Vérification de l'accord grammatical au sein d'un SN

Ainsi, ce graphe SN reconnaîtra par exemple la séquence étiquetée

<les,le.det+def+p> <vauriens,vaurien.noun+hum+m+p>

et associera à son analyse la structure de traits suivante :

$$\begin{bmatrix} CAT : SN \\ head : vauriens \\ subcat : hum \\ gender : m \\ number : p \end{bmatrix}$$

En revanche, ce même graphe n'analysera pas la séquence :

<les,le.det+def+p> <vaurien,vaurien.noun+hum+m+s>

puisque les valeurs *p* et *s* pour le trait *number* sont incompatibles et ne peuvent pas s'unifier.

contraintes existentielles

Les contraintes existentielles sont une autre forme d'équation sur les traits qui peuvent décorer une grammaire DRTN. Ce type d'équation permet d'imposer la présence (ou l'absence) d'un attribut sans spécifier sa valeur.

Une contrainte existentielle a une des deux formes suivantes :

– *chemin*
– *˜chemin*

La première forme, constituée d'un chemin fonctionnel, spécifie que l'attribut identifié par ce chemin doit être présent dans la structure de traits à l'issue de l'analyse. La seconde forme est une contrainte d'*inexistence* (préfixée par le symbole ˜) et spécifie que l'attribut doit être absent de la structure de traits.

Ainsi, par exemple, supposons que nous souhaitons ajouter la description des constructions avec pronominalisation clitique du complément d'objet direct dans notre grammaire du verbe *donner*. Nous pouvons représenter élégamment ces transformations en utilisant les contraintes existentielles.

La figure 4.27 présente le nouveau graphe V qui décrit le noyau verbal de

la phrase et dans lequel nous avons rajouté la possibilité de reconnaître un pronom clitique accusatif à gauche du verbe conjugué ; dans ce cas, nous unifions l'attribut *ppvacc* du non terminal V avec la forme fléchie du pronom clitique (équation *ppvac=$$.form*).

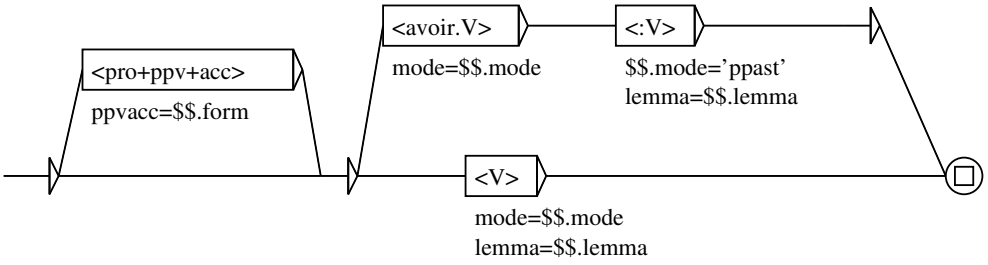

FIG. 4.27 – non terminal V modifié

Nous modifions également le graphe N0escroquerN1aN2 de manière à décrire ces nouvelles constructions (figure 4.28) : nous ajoutons en parallèle, à la reconnaissance de l'argument N1 en position d'objet direct, un nouveau chemin étiqueté en entrée par le mot vide contenant les deux équations :

n1=V.ppvacc
n1

La première équation unifie l'attribut *n1* associé au constituant P avec l'attribut *ppvacc* associé au noyau verbal de la phrase. Ce trait a pour valeur la forme fléchie du pronom clitique accusatif lorsqu'il est présent (figure 4.27) et a une valeur non spécifiée dans le cas contraire. La deuxième équation existentielle *n1* impose que le trait *n1* existe et que sa valeur soit spécifiée ; en d'autres termes, la seconde équation permet de s'assurer de la présence du pronom clitique dans la phrase. Nous procédons de la même manière pour décrire les *constructions alternées* où l'argument N2 prend la position d'objet direct.

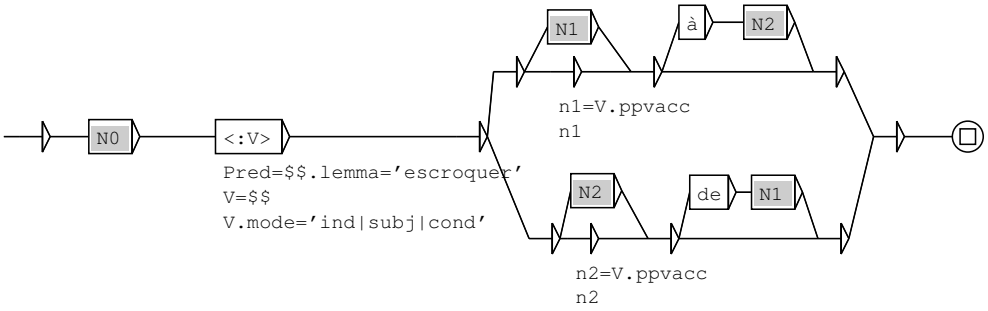

FIG. 4.28 – Nouveau graphe N0escroquerN1aN2

La grammaire ainsi modifiée analysera de nouvelles constructions telles que

Le fisc les a escroqués d'un sacré pactole
(ces 3 000 francs,) *Luc les a escroqués au fisc*

avec, comme résultat de l'analyse de la première phrase, l'arbre syntaxique et la structure de traits représentés par les figures suivantes :

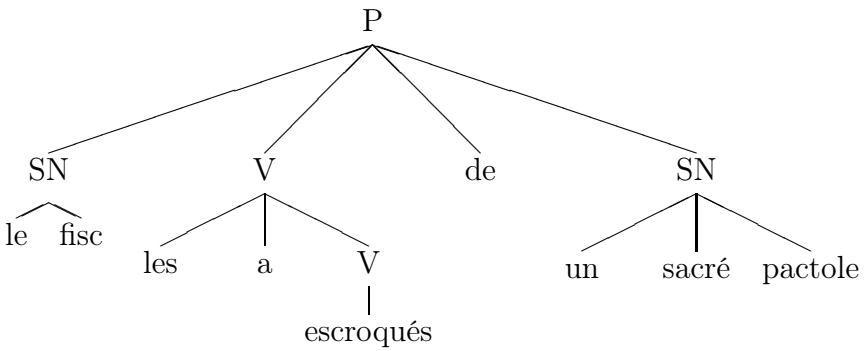

FIG. 4.29 – Arbre syntaxique pour la phrase *Le fisc les a escroqués d'un sacré pactole*

$$\begin{bmatrix} Pred : (1)escroquer \\ V : \begin{bmatrix} lemma : (1) \\ mode : ind \\ ppvacc : (2)les \end{bmatrix} \\ n0 : fisc \\ n1 : pactole \\ n2 : (2) \end{bmatrix}$$

FIG. 4.30 – Structure de traits obtenue suite à l'analyse de la phrase

Equation ensembliste

Les équations ensemblistes forment le troisième et dernier type d'équation pouvant décorer une grammaire DRTN. Ce type d'équation permet de construire des structures de traits à valeur ensembliste (cf. section 4.3.1), c'est-à-dire des attributs ayant pour valeur un ensemble de structures de traits. Une équation ensembliste à la forme suivante

$$chemin_1 < chemin_2$$

où $chemin_1$ et $chemin_2$ sont des chemins fonctionnels, et le trait désigné par $chemin_2$ a une valeur ensembliste. Une équation de cette forme permet de rajouter la valeur de l'attribut désigné par $chemin_1$ à l'ensemble de valeurs de l'attribut désigné par $chemin_2$.

Nous utilisons les équations ensemblistes pour construire des structures contenant un ensemble d'éléments dont on ne connaît pas a priori le nombre. Par exemple, la conjonction *et* permet de coordonner un ensemble de noms dont le nombre est a priori non borné au sein d'un même syntagme nominal :

Claudia et Takuya
Claudia, Mariana et Takuya
Claudia, Mariana, Lidia et Takuya
etc.

Nous pouvons facilement analyser ce type de constructions à l'aide des équations ensemblistes comme le montre le graphe de la figure 4.31.

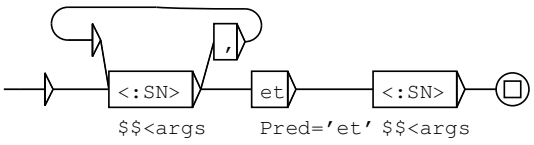

FIG. 4.31 – Représentation des SN coordonnés

Ce graphe analysera par exemple le syntagme nominal

Vincent, François, Paul et les autres

en produisant la structure de traits suivante :

$$\left[\begin{array}{l} Pred : et \\ args :< [\ head : Vincent\ ], [\ head : François\ ], [\ head : Paul\ ], [\ head : autres\ ] > \end{array}\right]$$

Raccourcis

Comme nous l'avons vu avec notre grammaire d'exemple, beaucoup des contraintes d'unification sont utilisées dans le but de faire remonter un attribut hérité d'un des sous-constituants au niveau du constituant englobant de manière à faire remonter des informations à des niveaux supérieurs dans l'arbre syntaxique. Ce type d'équation a toujours la même forme :

attribut=$$.attribut

(voir par exemple les équations *gender=$$.gender*, *number=$$.number* et *subcat=$$.subcat* dans le graphe SN de la figure 4.26).

Pour des raisons de facilités d'écriture et de lecture des grammaires, ainsi que de paresse, ce type d'équation peut être écrite de façon abrégée sous la forme suivante :

^attribut

Le graphe SN de la figure 4.26 peut être ainsi considérablement allégé en utilisant ce type de notation comme le montre la figure 4.32.

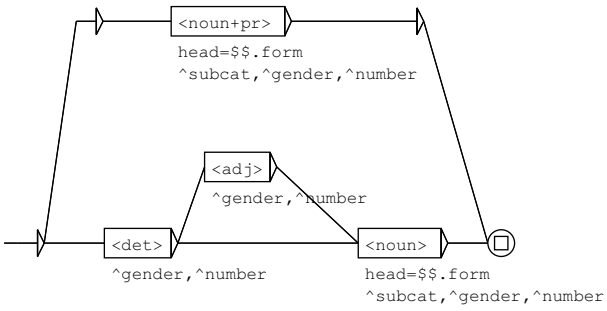

FIG. 4.32 – Utilisation des raccourcis d'écriture dans les contraintes d'unification

Enfin, l'équation ^ *$$* permet d'unifier dans sa totalité la structure de traits associée au symbole qui étiquette la boîte sous laquelle se trouve l'équation avec la structure de traits associée au constituant courant. Dans la pratique, nous n'avons pas utilisé ce type de contrainte durant l'écriture de notre grammaire. Cependant, les contraintes de cette forme peuvent certainement s'avérer utiles pour l'écriture d'autres grammaires dans le formalisme DRTN. Les grammaires LFG utilisent d'ailleurs intensivement des équations fonctionnelles à la sémantique équivalente (de la forme $\uparrow=\downarrow$).

4.4.1 Comparaison avec d'autres formalismes

Dans cette section, nous situons le formalisme des RTN décorés par rapport à d'autres formalismes grammaticaux existants qui ont été ou sont encore utilisés à des fins d'analyse syntaxique du français.

Grammaires locales

Les grammaires locales sont bien adaptées pour l'analyse de sous-langages spécifiques et pour l'analyse superficielle de textes en décrivant des phénomènes de micro-syntaxe (reconnaissance de groupes nominaux non récursifs, d'entités nommées, etc.). Dans [Paumier, 2003a], Sébastien Paumier montre

qu'il est également possible d'utiliser ce formalisme simple pour l'analyse syntaxique à large couverture. Cependant, ce travail s'est limité à l'analyse des phrases simples, et ne traite pas le problème des phrases enchassées (complétives, relatives, participiales, subordonnées circonstancielles, etc.). De plus, les grammaires locales ne permettent pas de formaliser de façon simple les phénomènes d'accords grammaticaux entre les constituants. Le travail présenté ici poursuit les recherches entreprises par S. Paumier. Tous ces problèmes non traités nous ont amené à enrichir le formalisme des grammaires locales par des contraintes d'unification afin de pallier toutes ces limitations.

LFG

Le formalisme des RTN décorés et celui de LFG sont très proches ; la principale différence étant qu'avec les RTN décorés, les règles de réécriture sont décrites sous la forme d'automates contrairement aux règles de réécriture hors-contexte utilisées dans LFG. L'utilisation d'automates nous permet de mettre aisément en relation (puisque dans le même graphe) différentes variantes syntaxiques considérées comme syntaxiquement équivalentes, telles que l'alternance sur la position des arguments par exemple.

Une autre différence fondamentale entre les deux approches est qu'avec LFG, les propriétés de sous-catégorisation et transformationnelles sont codées dans le lexique et non dans la grammaire. Ainsi, les règles de réécriture qui constituent une grammaire LFG sont typiquement générales et peu nombreuses ; la grammaticalité des phrases ainsi analysées est validée dans une seconde passe en testant la validité des structures fonctionnelles qui ont été construites à partir des stuctures qui sont associées à chaque élément du lexique présent dans la phrase à partir de règles de bonnne formation. Notre approche est différente dans le sens où notre grammaire est fortement lexicalisée et que nous décrivons explicitement, pour chaque élément prédicatif, quelles sont les possibles réalisations syntaxiques des constituants dans lesquels ils peuvent apparaitre.

Enfin, le formalisme LFG est avant tout un formalisme linguistique, dans lequel les structures de traits doivent être sous une certaine forme pour être considérées comme bien formées. Le formalisme de RTN décorés, en contre-

partie, est avant tout un outil formel qui ne spécifie aucune contrainte sur la constitution des structures de traits. Nous nous en sommes servi à des fins d'analyse syntaxique, mais ce même formalisme peut être utilisé pour d'autres types d'applications. Il a d'ailleurs déjà été utilisé à des fins d'extraction d'information [Watrin, 2006].

TAG

Le modèle des grammaires d'arbres adjoints (ou TAG) a été initialement défini dans [Joshi *et al.*, 1975], puis dans une version plus récente intégrant les structures de traits et l'unification dans [Vijay-Shanker, 1987]. Anne Abeillé a entrepris, dans le cadre de sa thèse [Abeillé, 1991], la construction d'une grammaire d'arbres adjoints pour le français en utilisant les ressources linguistiques élaborées aux LADL. Ce travail de description continue depuis plus de 15 ans pour donner lieu à une grammaire à large couverture d'une finesse de précision pour de nombreux phénomènes syntaxiques [Abeillé, 2002], qui est encore enrichie aujourd'hui [Barrier, 2006]. Les grammaires TAG appartiennent à la famille des grammaire d'unification [Abeillé, 1993] mais ce distinguent des autres grammaires de ce type en ce que les unités sont des arbres et non des règles de réécriture hors-contexte. Ces arbres se combinent entre eux à l'aide d'opérations de substitution et d'adjonction.

Le résultat de l'analyse d'un énoncé par une grammaire TAG se présente sous la forme d'un arbre dérivé et d'un arbre de dérivation. Ce dernier fournit une représentation sémantique des énoncés analysés, dans lequel apparaissent les relations prédicat-argument (obtenus par substitution) et modifieur-modifié (obtenu par adjonction). La forme d'un arbre pour représenter les résultat d'analyse est plus limitée qu'une représentation sous la forme d'une structure de traits puisqu'elle ne permet pas de représenter les relations de coréférences (par exemple lorsque un même complément est l'argument de plusieurs prédicats), ce qui peut se représenter à l'aide de structure de traits à valeur partagée.

De plus, l'adjonction permet de représenter de manière simple les verbes à complétive et les dépendances à distance, cependant, avec ce type d'analyse, les verbes à complétive sont considérés comme modifieurs de la phrase

subordonnée.

Ainsi, si nous considérons les séquences suivantes :

Jean pense que Marie partira
Jean pense partir
Jean pense à son départ

Ces trois séquences recevront les arbres de dérivations suivants à la suite de leur analyse par la grammaire TAG du français telle que présentée dans [Abeillé, 2002] [1] :

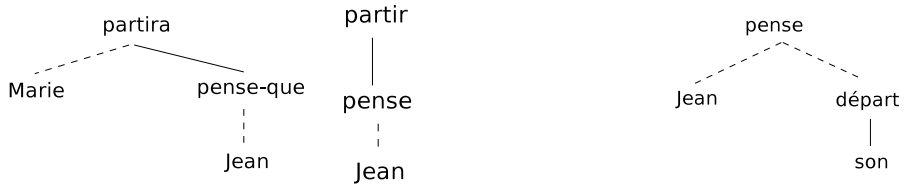

FIG. 4.33 – Arbres de dérivation obtenus après l'analyse TAG

Avec cette analyse, lorsque le complément du verbe *penser* a la forme d'une proposition complétive ou infinitive, ce verbe est considéré comme un modifieur de la proposition subordonnée, ce qui n'est pas le cas lorsque son complément à la forme d'un syntagme nominal. Nous ne sommes pas vraiment en accord avec ce type d'analyse (qui a néanmoins l'avantage d'être commode, puisque l'utilisation d'arbres auxiliaires pour représenter les verbes à compléments phrastiques permet de formaliser de façon très simple les phénomènes d'extraction et les dépendances à distance). En effet, nous pensons que le verbe principal *penser* entretient le même type de relation avec ces deux arguments dans les trois phrases de notre exemple. C'est pourquoi nous associons à ces trois séquences, des analyses dans lesquelles ce verbe est considéré comme le prédicat principal à deux arguments, le second ayant la forme d'une complétive, d'une infinitive ou d'un syntagme nominal (cf. figure 4.34).

[1] Les arcs en pointillés représentent les relations entre un prédicat et son argument (obtenus par subsitution) et les arcs plein les relations entre un modifié et son modifieur (obtenu par adjonction).

$$\left[\begin{array}{l} Pred : penser \\ n0 : Jean \\ n1 : \left[\begin{array}{l} Pred : partir \\ n0 : Marie \end{array}\right] \end{array}\right] \quad \left[\begin{array}{l} Pred : penser \\ n0 : (1)Jean \\ n1 : \left[\begin{array}{l} Pred : partir \\ n0 : (1) \end{array}\right] \end{array}\right] \quad \left[\begin{array}{l} Pred : penser \\ n0 : Jean \\ n1 : [\ Pred : départ\ ] \end{array}\right]$$

FIG. 4.34 – Analyses correspondantes par grammaire DRTN

4.5 Grammaire du Français

Nous avons ainsi travaillé sur la construction d'une grammaire DRTN lexicalisée du français en exploitant les informations de sous-catégorisation et transformationnelles des prédicats du français décrits dans les tables du lexique-grammaire. Notre grammaire est lexicalisée dans le sens où, pour chacun de ces éléments (verbe plein, nom ou adjectif prédicatif ou expression figée), nous décrivons explicitement dans notre grammaire l'ensemble des réalisations possibles de chaque constituant syntaxique dont il peut être le noyau.

Ces constituants peuvent être des phrases complètes, des propositions infinitives, participiales, des propositions relatives dans lesquelles un élément a été extrait, des groupes nominaux complexes dont la tête est un nom prédicatif, etc.

Comme nous le verrons, nous avons pu réduire la complexité d'une telle description systématique par l'écriture manuelle d'une meta-grammaire à partir de laquelle nous générons l'ensemble de ces descriptions lexicalisées en fonction du contenu des tables du lexique-grammaire.

4.5.1 Construction semi-automatique

Notre grammaire DRTN lexicalisée finale, qui est utilisée pour l'analyse syntaxique, est composée de l'ensemble des grammaires spécialisées pour chacun des éléments prédicatifs que nous traitons. Cette grammaire est générée automatiquement à partir d'un ensemble de meta-grammaires que nous avons

construites manuellement. Une meta-grammaire est composée d'un ensemble de graphes paramétrés associés à une table du lexique-grammaire. L'utilisation des graphes paramétrés fonctionne sur le même principe que le système de génération de grammaires par graphes patrons inclus dans les logiciels INTEX et Unitex (même si la nomenclature est un peu différente). Ce système de graphe patron a été repris dans plusieurs travaux exploitant les tables du lexique-grammaire pour l'analyse de textes ([Senellart, 1999] [Paumier, 2003a] par exemple) ou des recherches linguistiques sur corpus [Camugli Gallardo et Blanc, 2006].

Dans le cas de notre grammaire, chaque graphe paramétré décrit un constituant syntaxique de la grammaire dont le prédicat est une variable qui sera instanciée durant la phase de lexicalisation de la grammaire. Nous pouvons voir la meta-grammaire d'une table (c'est-à-dire l'ensemble des graphes associés à cette table) comme la grammaire de toutes les constructions syntaxiques de cette table, même si chaque construction n'est pas compatible avec toutes les entrées. Chaque chemin de cette grammaire est identifié par un paramètre qui réfère à la propriété qui lui correspond dans la table. Un paramètre a le format suivant : @X@, où X est l'intitulé d'une colonne de la table. Durant la phase de lexicalisation, le processus de génération de la grammaire construit pour chaque entrée de la table, une grammaire spécialisée dans laquelle seuls les chemins correspondant aux propriétés syntaxiques acceptées par cette entrée sont conservés. Lorsqu'une propriété identifiée par un paramètre est acceptée par l'entrée (c'est-à-dire lorsque l'intersection de la ligne et de la colonne considérées contient un $+$), le paramètre est remplacé par la chaîne vide dans le grammaire générée. La chaîne vide assure la continuation du chemin dans l'automate et donc que la construction est conservée. Quand la propriété n'est pas acceptée, la transition est supprimée de la grammaire ce qui a pour effet de bloquer l'ensemble des chemins passant par cette transition. Certaines propriétés ne sont pas de nature booléenne, mais contiennent des valeurs textuelles (par exemple la valeur d'une préposition), dans ce cas, le paramètre est remplacé dans la grammaire lexicalisée par la valeur de cette propriété. Il est également possible de nier un paramètre : le paramètre @!X@ signifie que le chemin identifié par ce paramètre sera conservé uniquement si la propriété d'intitulé X n'est pas acceptée par l'entrée considérée. Enfin, le paramètre spécial @%id@ sera remplacé au moment de la lexicalisation par le numéro de ligne de l'entrée considérée. Ce paramètre est utile pour identifier de façon unique une entrée d'une table du

lexique grammaire, même lorsque cette table contient plusieurs entrées pour un même lemme.

Par exemple, la figure 4.35 présente un graphe paramétré associé à la table 6. Cette table comporte la description des verbes à structure transitive simple avec complément direct phrastique, c'est-à-dire les verbes (comme *admettre*) qui entrent dans la construction N0 V N1, avec N1=:que P + N :

Luc a finalement admis (son erreur + que Lea l'a quitté).

Nous décrivons dans notre graphe les formes de base pour les verbes de cette table[2]. Les différents chemins de la grammaire sont identifiés par des paramètres qui réfèrent aux propriétés de la table leur correspondant. Par exemple, le paramètre @N0=Nhum@ réfère à la colonne dans la table indiquant si le verbe admet un sujet humain, le paramètre @N0V@ réfère à la colonne qui indique si le verbe admet l'ellipse de son objet direct. De même, le paramètre @entry@ à valeur textuelle fait référence à la colonne dans la table où est donné le lemme de l'entrée. Le graphe de la figure 4.36 montre le résultat de la lexicalisation de la grammaire pour le verbe *admettre*.

Contrairement à ce que laisse penser notre exemple, nous ne construisons pas, durant la lexicalisation de la meta-grammaire d'une table, un graphe lexicalisé différent pour chaque combinaison de graphe paramétré et d'entrée de la table. Dans la pratique, nous aplatissons au préalable chaque graphe paramétré principal décrivant un constituant syntaxique de la grammaire, de manière à ce qu'il soit décrit par un unique automate sans appel à des sous-graphes (cf. section 3.4). Puis, à partir de chacun de ces automates paramétrés, nous construisons sa version lexicalisée constituée de l'union des descriptions de ce constituant syntaxique lexicalisées pour chacune des entrées de la table. La grammaire générale est ensuite obtenue en construisant chaque constituant syntaxique par l'union des constituants lexicalisés de chaque table. Nous optimisons également notre grammaire, en effectuant

[2]Dans la pratique, nous avons fait la description du verbe principal et de ses arguments N0 et N1 dans des sous-graphes distincts de manière à pouvoir les réutiliser directement pour la description d'autres types de constructions. Nous avons inclus ces descriptions dans le graphe principal pour notre exemple de manière à illustrer le procédé de lexicalisation de la meta-grammaire.

pour chacun de ces constituants, les opérations d'émondation, de suppression des transitions étiquetées par le mot vide, de déterminisation et de minimisation telles que nous les appliquons pour l'optimisation des grammaires WRTN (cf. section 3.4).

Notons que nous aurions pu mettre en place un système permettant de générer, à partir d'une unique meta-grammaire, l'ensemble des constructions lexicalisées de chaque entrée prédicative décrite dans les tables du lexique-grammaire. C'est d'ailleurs l'approche qui a été choisie dans [Paumier, 2003a], par l'utilisation d'une super table. Cependant, un tel procédé n'est pas trivial à mettre en place puisqu'il existe de nombreuses propriétés qui sont spécifiques à certaines tables uniquement. Ainsi, le fait de représenter toutes ces constructions dans une unique meta-grammaire alourdirait considérablement cette dernière au risque de la rendre difficile à lire et à maintenir. De plus, certaines propriétés peuvent avoir un même intitulé mais une signification différente en fonction de la table dans laquelle elle sont encodées[3]. Pour toutes ces raisons, nous avons décidé de construire manuellement une meta-grammaire différente pour chaque table. Après avoir expérimenté avec ce système, nous nous sommes aperçu que nous obtenons de nombreuses redondances dans les descriptions des différentes meta-grammaires, ce qui cause également des problèmes pour la maintenance de la grammaire dans son ensemble. Nous en sommes donc arrivé à la conclusion qu'il serait sans doute souhaitable de faire un compromis entre ces deux approches, en mettant en place un système, que nous n'avons pas eu le temps de développer, dans lequel une meta-grammaire serait associée à un ensemble de tables qui auraient certaines propriétés en commun, comme par exemple le nombre des actants acceptés par les prédicats.

[3]Ce problème n'en est plus un aujourd'hui, puisqu'une révision complète des intitulés des colonnes a été effectuée récemment.

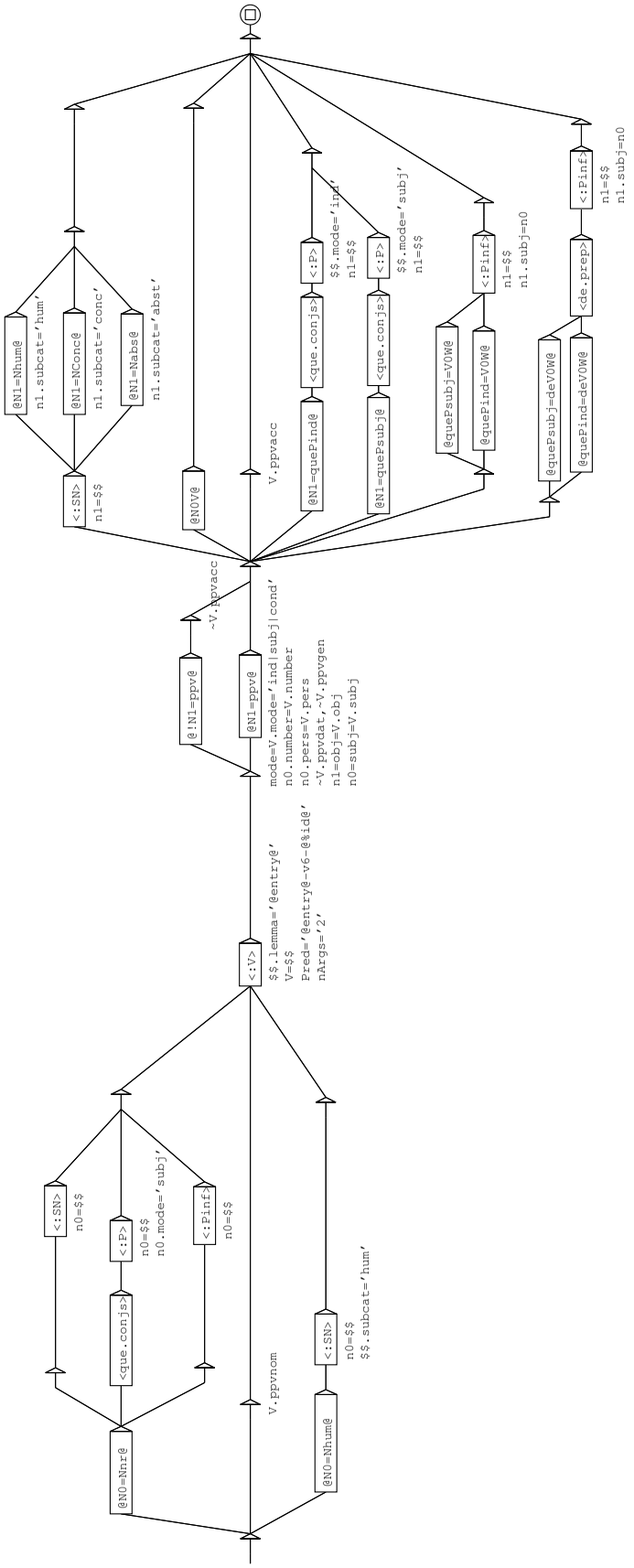

FIG. 4.35 – Exemple de graphe paramétré

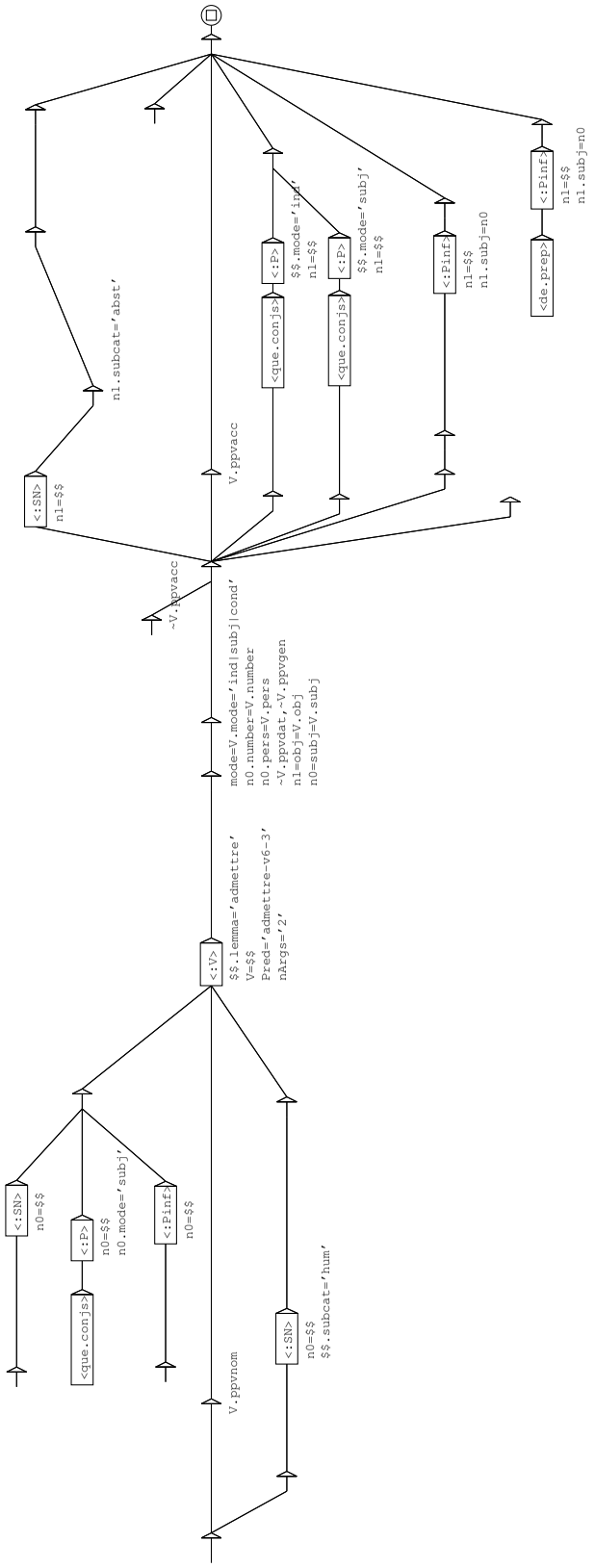

FIG. 4.36 – Spécialisation pour le verbe admettre

4.5.2 Remarques sur l'analyse

Nous avons implémenté un nouvel analyseur spécialisé pour les grammaires d'unification DRTN. Théoriquement, nous aurions pu implémenter notre analyseur DRTN comme une routine de post-traitement branchée à notre analyseur pour grammaire WRTN présenté dans le chapitre précédent. Cette routine aurait été appelée après l'analyse grammaticale par la grammaire WRTN support, et aurait vérifié les contraintes d'unification et construit les structure de traits résultat à partir de la forêt d'arbres décorés obtenue comme résultat de l'analyse. C'est d'ailleurs cette approche qui a été choisie pour l'implémentation de l'analyseur LFG efficace SxLfg [Boullier et Sagot, 2005] qui fonctionne en deux passes : une première passe effectue la reconnaissance par la grammaire algébrique support et une seconde passe vérifie les contraintes d'unification sur les arbres de dérivation obtenus. Cependant, cette approche ne nous a pas semblé viable dans notre cas, du fait de la forme de notre grammaire qui est fortement lexicalisée au niveau des contraintes d'unification. En raison des ambiguïtés, la grammaire contient un grand nombre de chemins qui reconnaissent les mêmes séquences en entrées et qui diffèrent seulement par les contraintes d'unification spécialisées pour chaque entrée lexicale. Ainsi, nous avons adapté notre algorithme d'analyse pour les grammaires DRTN de manière à vérifier les contraintes d'unification présentes dans les sorties et à construire les structures de traits associées à chaque constituant syntaxique durant l'analyse. Ceci nous permet de stopper plus tôt la reconnaissance d'analyses qui échouent du fait que les structures de traits qui leur sont associées ne peuvent pas s'unifier. Nous obtenons, comme résultat de notre analyseur, une forêt d'arbres de dérivation ; à chacun de ces arbres est associée une structure de traits bien formée. L'arbre syntaxique rend compte du découpage syntagmatique de la phrase et nous réprésentons dans la structure de traits associée différentes relations grammaticales qui relient ces constituants.

Dans le cas de notre grammaire, nous nous sommes essentiellement intéressé à identifier dans les textes les prédicats sémantiques et leurs arguments. Nous avons ainsi adopté une convention de notation particulière pour représenter ce type de relations dans la structure de traits. Les prédicats sont représentés par un trait `Pred` ; la valeur de ce trait identifie de façon unique une entrée d'une table du lexique-grammaire : celle-ci consiste en la concaténation du

lemme de l'entrée, suivi du nom de la table à laquelle elle appartient suivi par son identifiant dans cette table (i.e. le numéro de la ligne dans laquelle l'entrée est décrite). Par exemple, la valeur du trait `Pred` pour l'emploi du verbe *admettre* décrit dans la table 6 est `admettre-v6-3` (cf. l'exemple de la figure 4.36). Nous accompagnons l'attribut `Pred` par un attribut `nArgs` qui doit être présent au même niveau dans la structure de traits. La valeur de cet attribut indique le nombre d'arguments attendus par ce prédicat (voir l'équation `nArgs='2'` dans notre exemple de la figure 4.36). Les valeurs de ces arguments sont stockées, lorsqu'ils sont présents, dans les traits `n0`, `n1`, etc.

Nous représentons également certaines relations de modifieur à modifié dans notre grammaire. Par exemple, nous analysons une phrase subordonnée comme un modifieur de la phrase principale, ou encore une phrase relative ou participiale comme modifieur du syntagme nominal auquel elle est accolée. Tous ces modifieurs sont stockés dans l'attribut `modifs` de la structure de traits décrivant le constituant qu'ils modifient. Cet attribut prend pour valeur un ensemble de structures de traits.

Par exemple, étant donnée la phrase suivante :

L'homme qui parlait à Lea est parti.

Nous obtenons comme résultat de son analyse avec notre grammaire, la structure de traits suivante, que nous avons simplifiée en conservant uniquement les informations qui nous intéressent :

$$\left[\begin{array}{ll} Pred : \text{partir-v2-61} & \\ nArgs : 3 & \\ n0 : \left[\begin{array}{ll} head : (1)\left[\ form : homme\ \right] & \\ modifs :< \left[\begin{array}{l} Pred : \text{parler-v15-45} \\ nArgs : 3 \\ n0 : (1) \\ n2 : \left[\ head : \left[\ form : Lea\ \right]\ \right] \end{array}\right] > & \end{array}\right] & \end{array}\right]$$

FIG. 4.37 – Structure de traits (simplifiée) résultat de l'analyse

Notons que la valeur de l'attribut `nArgs` ne correspond pas au nombre d'arguments essentiels qui sont effectivement présents dans l'énoncé analysé ; cette valeur correspond au nombre d'arguments qu'accepte le prédicat dans la phrase canonique qui a servi à sa classification dans les tables du lexique-grammaire (dans laquelle tous ses arguments jugés comme essentiels sont réalisés). Par exemple, le prédicat principal de notre phrase analysée a été identifié comme étant le verbe *partir* dans son emploi défini dans la table 2. Ce verbe accepte dans sa forme la plus longue trois arguments, dont un complément locatif et une proposition infinitives qui peuvent être réalisés tous les deux dans le même énoncé :

Luc est parti à la boulangerie chercher du pain.

Dans notre exemple, seul l'argument `n0` (le sujet) est réalisé syntaxiquement.

De manière à faciliter la lecture des résultats d'analyse, nous avons implémenté un extracteur de relations entre prédicat et arguments. Etant donnée une structure de traits dont la forme suit nos conventions de notation, notre programme en extrait, par un simple parcours recursif de la structure, toutes ces relations et les présente sous la forme de prédicats logiques. Ainsi, nous obtenons comme résultat de l'analyse de notre phrase les deux formules suivantes :

partir-v2-61(homme, _, _)
parler-v15-45(homme, _, Lea)

4.5.3 Constituants syntaxiques généraux

Outre les descriptions lexicalisées, qui sont générées à partir de notre metagrammaire, nous avons également fait la description de constituants généraux non lexicalisés qui sont utilisés dans la plupart des constituants de notre grammaire. Il s'agit, entre autres, des constituants `Ins`, pour la description des adverbes d'une phrase, `SN` pour la description des syntagmes nominaux, `SA` pour celles des syntagmes adjectivaux, et `V` pour la description du noyau verbal. Notons que ces descriptions générales peuvent également invoquer des

descriptions lexicalisées. Par exemple, notre grammaire des `SN` reconnaît des formes générales de groupes nominaux, mais intègre également les descriptions lexicalisées des groupes nominaux à tête prédicative qui sont générées à partir de la meta-grammaire des tables de noms. Cette même grammaire décrit la possibilité de modifier le nom tête par une proposition relative qui est elle-même décrite par la grammaire lexicalisée de son prédicat principal. Il en est de même pour la grammaire des adverbes, qui comprend par exemple la description des phrases introduites par une conjonction de subordination. Ainsi, toutes ces descriptions générales forment une sorte de liant entre les descriptions lexicalisées décrites dans la meta-grammaire, et permettent de ce fait d'obtenir une grammaire cohérente capable d'analyser des énoncés complexes comprenant plusieurs propositions enchassées.

Nous présentons ici plus en détail notre grammaire `SN` décrivant les syntagmes nominaux et notre constituant `V` qui décrit le noyau verbal d'une phrase (conjuguée, infinitive ou participiale).

Constituant SN

Le graphe de la figure 4.38 présente notre grammaire `SNsimple`, pour la description des syntagmes nominaux simples, c'est-à-dire non coordonnées. A gauche, nous décrivons la tête du syntagme nominal, qui peut être réalisée soit par un pronom tonique (*moi*, *lui-même*, etc.), indéfini (*personne*, *quelqu'un*, etc.) ou démonstratif (*celui-ci*, *ceux*, etc.), soit par un nom propre décrit dans le constituant `Npr`, soit par un autre groupe nominal non récursif décrit dans le constituant `chunk`. Dans tous les cas, nous unifions la structure de traits associée à ce constituant avec l'attribut `head` du constituant englobant (avec l'équation `head=$$`), et nous faisons remonter les informations indiquant son genre, son nombre, sa personne, sa sous-catégorie sémantique et son lemme au niveau du `SN` (équations `^gender`, `^number`, `^pers`, `^subcat` et `^lemma`).

Le constituant `chunk` est présenté dans la figure 4.39 ; ce graphe décrit le syntagme nominal jusqu'à son nom tête (c'est-à-dire précédé d'un ou plusieurs déterminants et éventuellement d'un groupe adjectival), les éventuels modifieurs à droite de celui-ci sont décrits dans le graphe principal `SNsimple`. Nous ne présentons pas les graphes `Det` et `Adjg`, décrivant respectivement les dé-

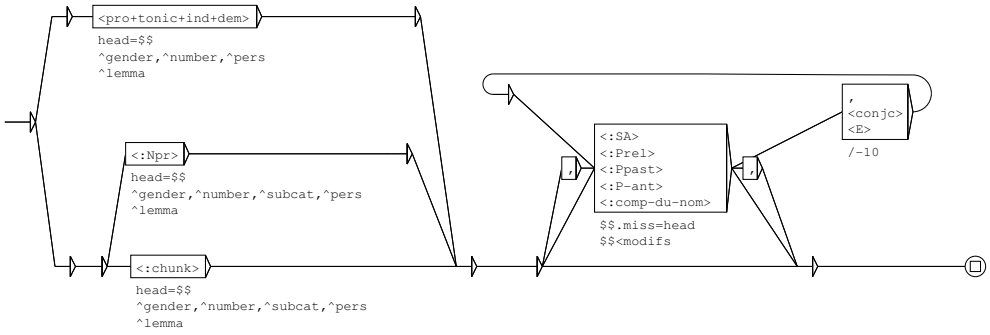

FIG. 4.38 – Constituant SNsimple

terminants et groupes adjectivaux qui peuvent apparaître à gauche du nom. Ces graphes sont pour l'instant très simples, nous avons cependant décrit ces éléments dans des sous-graphes distincts de manière à pouvoir facilement les remplacer par des descriptions plus complètes. La pondération de poids 10 dans notre grammaire sur la transition identifiant le nom tête comme un nom composé permet de préférer cette analyse aux analyses identifiant le mot de tête comme un mot simple (de poids 0). En effet, beaucoup de noms composés français (comme *pomme de terre* ou *carte bleue* par exemple) ont des structures internes (de type `NA` ou `NprepN`) qui les rendent difficiles à distinguer des mots simples suivis par un modifieur ; cette heuristique nous permet donc de réduire le nombre d'ambiguïtés d'analyse en préférant les analyses figées aux analyses compositionnelles.

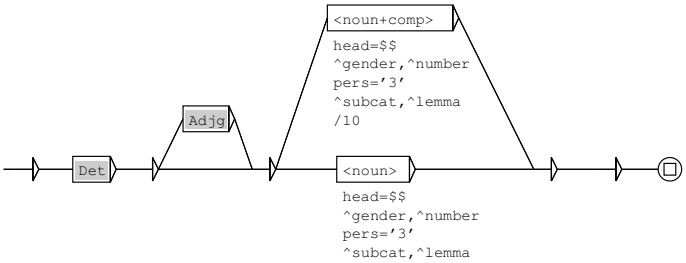

FIG. 4.39 – Constituant `chunk`

Dans la partie droite du graphe `SNsimple`, nous décrivons les modifieurs qui peuvent apparaître à droite du nom tête : il peut s'agir d'un syntagme adjectival (constituant `SA`), d'une phrase relative (constituant `Prel`), d'une participiale passé ou présent (constituants `Ppast` et `P-ant`) ou d'un complément du nom (constituant `comp-du-nom`). Nous ajoutons la structure de traits associée à ce constituant à la liste des modifieurs du syntagme nominal avec l'équation `$$<modifs`. Avec l'équation `$$.miss=head`, nous unifions le trait `miss` du modifieur avec la structure de traits associée à la tête de notre syntagme nominal. Nous utilisons ce type d'équation dans différentes parties de notre grammaire afin de décrire les différents phénomènes d'extraction en transmettant l'information sur l'élément manquant dans le constituant privé de cet élément par l'intermédiaire du trait `miss`.

Par exemple, la figure 4.40 présente notre graphe `Prel` décrivant les propositions relatives. Nous y décrivons, entre autres, la possibilité d'introduire la proposition relative par le pronom *qui* ou *lequel* (ou une de ses formes fléchies) ; dans ce cas, le pronom est suivi par une phrase privée de son sujet décrit par le constituant `P-Nnom`. L'équation `^miss` sous la boîte étiquetée par l'appel à `P-Nnom` permet de propager le trait `miss` dans la description de ce constituant. Notre grammaire `P-Nnom` est présentée dans la figure 4.41, ce graphe consiste simplement en des appels à des sous-graphes sur l'ensemble des grammaires lexicalisées pour le constituant `P-Nnom` que nous avons décrit dans notre meta-grammaire. Nous donnons des exemples de ces descriptions dans la section 4.5.4 qui présente des extraits de notre meta-grammaire.

Ce même trait pourra éventuellement être propagé dans d'autres propositions enchassées nous permettant ainsi de formaliser les dépendances à distance à la manière du trait *Slash* utilisé en HPSG. Par exemple, notre grammaire reconnaît, avec ce mécanisme de propagation, la séquence suivante comme un syntagme nominal :

> la fille que Luc dit avoir reconnue.

et lui associe l'analyse suivante, dans laquelle le nom *fille* antéposé, dont la structure de traits a été propagée dans les propositions enchassées par le biais du trait `miss`, a été identifié comme l'argument *n1* du verbe *reconnaître* :

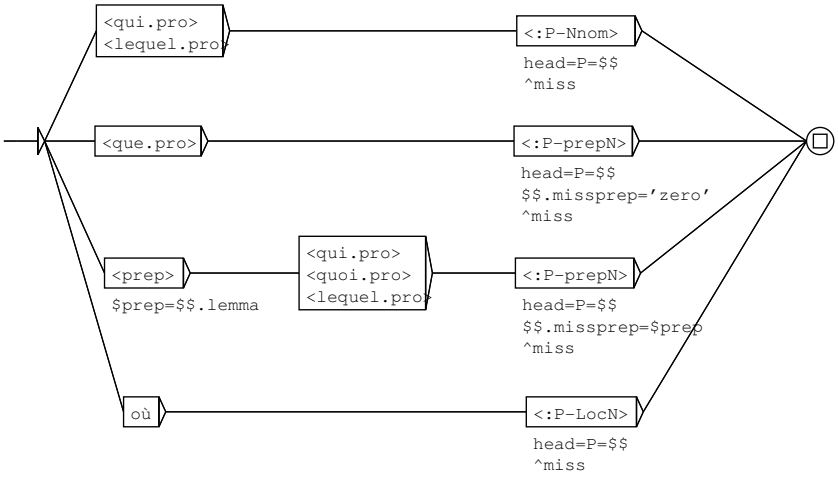

FIG. 4.40 – Constituant `Prel`

dire-v9-125(Luc, reconnaître-v32r2-352(Luc, fille), _)

Nous présentons, pour exemple, dans la figure 4.42 le graphe `comp-du-nom` qui décrit les compléments du nom. Ce graphe, très général, décrit ce type de modifieur comme un syntagme nominal introduit par une préposition. A l'aide des équations sur les traits, nous associons à ce type de construction une analyse dans laquelle le prédicat est la préposition introductrice qui accepte deux arguments qui sont respectivement le nom tête du syntagme nominal (unifié par l'intermédiaire de l'attribut `miss`) et le nom tête du complément du nom. Ainsi, par exemple, les syntagmes nominaux :

la pomme sur la table
le fils du facteur
les jardins de Paris
la boîte de trombonnes
la barre de métal

recevront les analyses :

FIG. 4.41 – Constituant `P-Nnom`

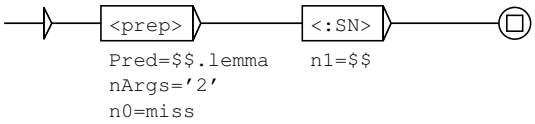

FIG. 4.42 – Constituant `comp-du-nom`

sur(pomme, table)
de(fils, facteur)
de(jardins, Paris)
de(boîte, trombonnes)
de(barre, métal)

Comme nous le voyons, une même préposition peut introduire différents types de relations entre le modifieur et son modifié ; nous n'avons pas cherché pour l'instant à affiner cette description, puisque nous n'avons pas les moyens de résoudre ce problème dans le cas général[4]. Cependant, dans le cas où le nom

[4]Afin de décider de la nature de la relation qui est introduite par la préposition, il nous semble nécessaire d'avoir à notre disposition une ontologie fine dans laquelle seraient classés les deux substantifs qui intéragissent dans cette relation.

tête du syntagme nominal est un nom prédicatif décrit dans notre meta-grammaire, nous pouvons obtenir des analyses plus fines dans lesquelles les compléments du nom sont reconnus comme des arguments du prédicat en fonction de leur préposition introductrice (cf. section 4.5.4).

Enfin, pour terminer la description de notre grammaire des SN, notons que nous avons décrit la possibilité de reconnaître un nombre non borné de modifieurs à droite du nom tête en bouclant sur ces descriptions dans notre grammaire (figure 4.38). Nous avons cependant pondéré cette boucle par un poids négatif de -10, de manière à réduire les ambiguïtés d'attachement prépositionnel. Cette pondération négative nous permet d'associer des poids moins élevés aux analyses dans lesquelles la distance entre le modifieur et son modifié est plus importante. Cette heuristique nous permet ainsi, dans le cas où le nom de tête du SN est suivi par une séquence de modifieurs, de préférer les analyses où chaque modifieur est attaché au nom le plus proche à sa gauche. Ainsi, si nous prenons la séquence suivante qui est naturellement ambiguë :

Le fils du facteur que Luc nous a dit avoir reconnu

Cette séquence reçoit à l'issue de l'analyse par notre grammaire une unique analyse sur les deux possibles, dans laquelle Luc reconnaît le facteur et non le fils de ce dernier :

de(fils, facteur)
dire-v9-125(Luc, reconnaître-v32r2-352(Luc, facteur), nous)

Nous avons recouru à cette heuristique pour des raisons purement pratiques ; en effet, sans celle-ci, l'ambiguïté d'attachement prépositionnel nous créait un nombre tellement important de résultats d'analyse (de l'ordre de mille analyses par phrase) que ces résultats étaient inexploitables en pratique. Nous avons ainsi préféré, quitte à perdre l'analyse correcte, limiter de manière arbitraire le nombre de résultats d'analyse pour les cas d'ambigïté que nous ne savons pas résoudre a priori. Nous avons néanmoins conservé la boucle dans notre grammaire, pour analyser des cas où un modifieur ne peut pas être attaché au nom le plus proche à sa gauche, comme par exemple dans la séquence :

d'après les conclusions de ce rapport, approuvées par le conseil européen

Ici, la phrase participiale de verbe *approuver* ne peut pas être attachée au nom *rapport* du fait que ces deux mots ne vérifient pas les contraintes d'accord en genre et en nombre spécifiées dans notre grammaire. Ainsi, nous obtenons à l'issue de l'analyse, le résultat suivant dans lequel l'argument `n1` du verbe *approuver* est bien le mot *conclusions* et non *rapport* :

de(conclusions, rapport)
approuver-v12-10(conseil européen, conclusions)

Non terminal V

Nous présentons dans la figure 4.43 le graphe principal du constituant syntaxique `V`, qui décrit le noyau verbal d'une phrase (conjuguée, infinitive ou participiale), c'est-à-dire son verbe principal éventuellement modifié par des auxiliaires de temps, modaux ou aspectuels. Nous décrivons également dans ce constituant la possibilité d'apparition de pronoms clitiques à gauche du verbe (par un appel au sous-graphe `Cl` présenté dans la figure 4.44).

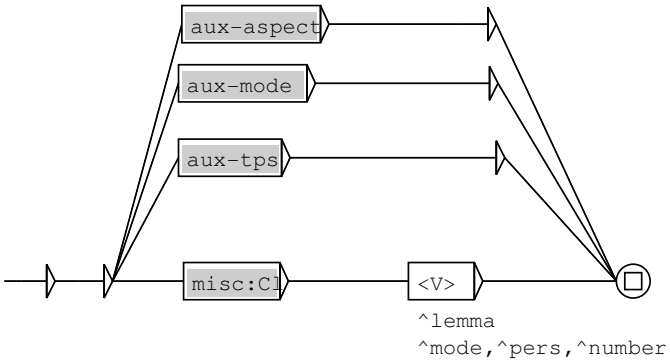

FIG. 4.43 – Constituant `V`

L'ordre d'apparition de ces pronoms est assez contraint en français, comme le montre notre description :

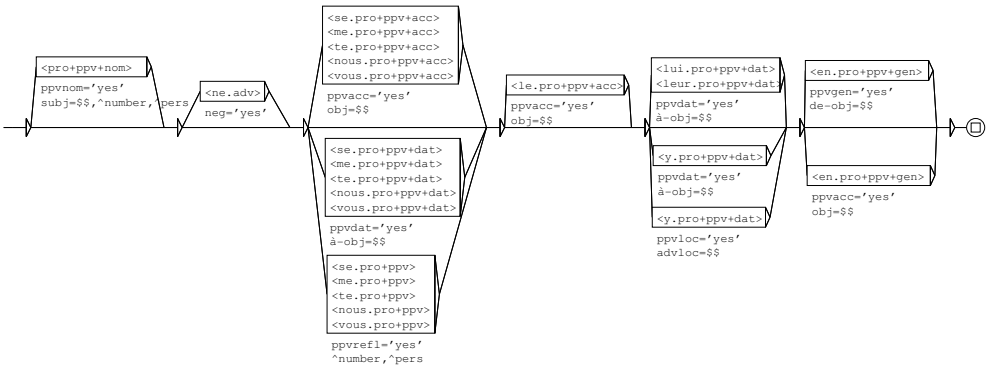

FIG. 4.44 – Sous-graphe `Cl`

– d'abord le pronom sujet (*je*, *tu*, *il*, etc.) ; dans ce cas, nous positionnons à *yes* la valeur du trait `ppvnom` du constituant `V` et nous unifions la valeur du trait `subj` avec la structure de trait associée au pronom sujet ; les équations `^pers` et `^number` permettent de vérifier l'accord en genre et en nombre entre le verbe et son sujet.
– il est ensuite possible de voir apparaître un des pronoms personnels *me*, *te*, *se*, *nous* ou *vous*. Le pronom peut recevoir alors 3 analyses distinctes :
  – soit il s'agit d'un pronom accusatif : *Luc nous observe*. Dans ce cas, nous positionnons la valeur du trait `ppvacc` à *yes* et nous unifions la structure de trait associée à ce pronom avec le trait trait `obj` du constituant `V` (équation `obj=$$`).
  – soit il s'agit d'un pronom datif : *Luc nous a offert un verre*. Dans ce cas, nous positionnons la valeur du trait `ppvdat` à *yes* et nous unifions la structure de trait associée au pronom avec le trait `à-obj` du constituant `V`.
  – soit il s'agit d'un pronom réfléchi : *Luc s'endort*. Dans ce cas, il faudrait distinguer quatre cas : le *se* figé (*Luc s'enquiert de la santé de Lea*) ; le *se* grammatical (*la branche se casse*) ; et le *se* résultat de la pronominalisation d'un complément direct (*Luc se prend pour Napoleon*) ou indirect (*Luc s'envoie un message sur son répondeur*). Nous ne faisons pas pour l'instant cette distinction dans notre grammaire et nous positionnons la valeur du trait `ppvrefl` à *yes* dans tous ces cas, en considérant que

le pronom réflechi fait parti du prédicat verbal. Nous distinguons de cette manière, le verbe intransitif *s'endormir* (décrit dans la table 35R) du verbe transifif *endormir* (table 4), prédicat de la phrase : *(Luc+le discours de Luc) nous endort.*

- les pronoms préverbaux *le*, *la*, *les* et les pronoms *lui*, *leur* ou *leurs* reçoivent la même analyse que les pronoms personnels respectivement accusatifs et datifs.
- le pronom préverbal *y* peut recevoir deux analyses : soit il pronominalise un complément datif : *Luc y pense tous les jours* ; soit il pronominalise un adverbe le plus souvent locatif : *le resto U, Luc y mange tous les jours.* Cet adverbe peut également être un argument essentiel d'un prédicat à complément locatif : *Luc y met son dentifrice.* Dans ce dernier cas, nous positionnons la valeur de l'attribut `ppvloc` à *yes* et nous unifions le trait `advloc` avec la structure de traits associée à ce pronom.
- Enfin, la forme *en* peut également recevoir deux analyses : il peut pronominaliser un complément introduit par la préposition *de* (*Luc en parle*), mais également un complément accusatif comme dans la phrase *Luc en a déjà mangé.*

Dans sa forme la plus simple, un noyau verbal consiste en un verbe éventuellement précédé par des pronoms clitiques (cf. figure 4.43). Dans ce cas, nous faisons remonter au niveau du constituant `V`, les informations sur le lemme du verbe, son mode, sa personne et son nombre ainsi que les analyses sur les pronoms clitiques comme vu précédémment.

Nous décrivons dans des sous-graphes, la possibilité de modifier le verbe par un auxiliaire de temps (sous-graphe `aux-tps`, figure 4.45), un auxiliaire de mode (graphe `aux-mode`, figure 4.46) ou un auxiliaire aspectuel (graphe `aux-aspect`, figure 4.47).

Tous ces graphes ont la même forme générale : ils décrivent l'auxiliaire suivi par un constituant `V` soit au participe passé pour les auxiliaires de temps *être* et *avoir* soit à l'infinitif pour les autres auxiliaires. Le mode du constituant `V` courant est hérité du mode de conjugaison de l'auxilaire, il en est de même pour les informations de nombre et personne (équations `^mode,^number,^person`). Le lemme du verbe principal, quant à lui, est hérité du constituant `V`, fils du constituant courant ; il en est de même pour les analyses issues de la présence de pronoms clitiques (excepté pour le pronom

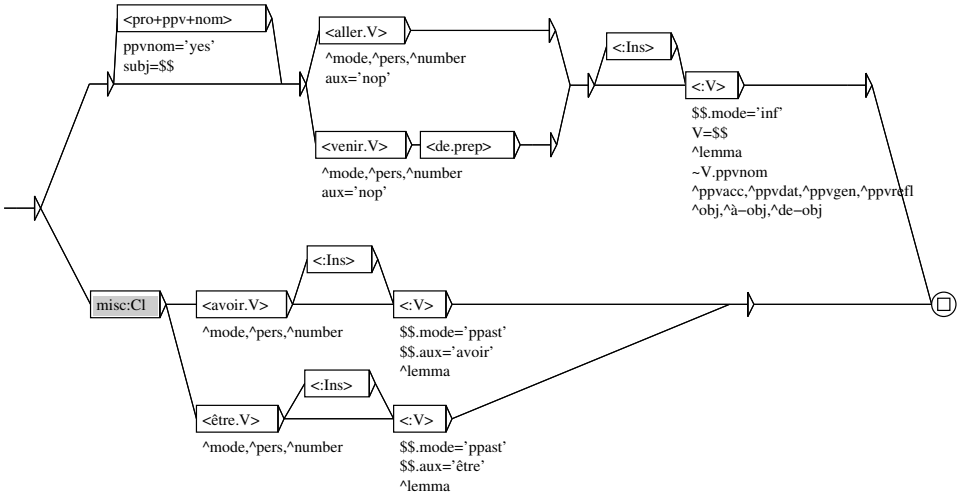

FIG. 4.45 – Sous-graphe `aux-tps`

sujet qui ne peut être réalisé qu'à gauche du verbe auxiliaire). Enfin, la reconnaissance récursive d'un constituant `V` à droite de l'auxiliaire, nous permet d'analyser des constructions où le verbe principal est modifié par plusieurs auxiliaires comme la séquence :

(Luc) semble avoir été en train de travailler

qui est analysée par notre grammaire comme un constituant `V` de lemme *travailler* et conjugué à l'indicatif.

La liste des verbes auxiliaires que nous avons décrits dans notre grammaire ne prétend pas être exhaustive ; en fait, nous nous sommes contenté pour l'instant de reprendre les exemples de ce type de verbes utilisés pour la présentation de la grammaire LTAG pour le français [Abeillé, 2002]. Il s'agit essentiellement de verbes décrits dans la table 1 du lexique-grammaire, c'est-à-dire les verbes à un complément de la forme d'une proposition infinitive qui n'a pas de source complétive attestée. Toutes les entrées de cette table ne peuvent pas pour autant être considérées comme des auxiliaires modaux

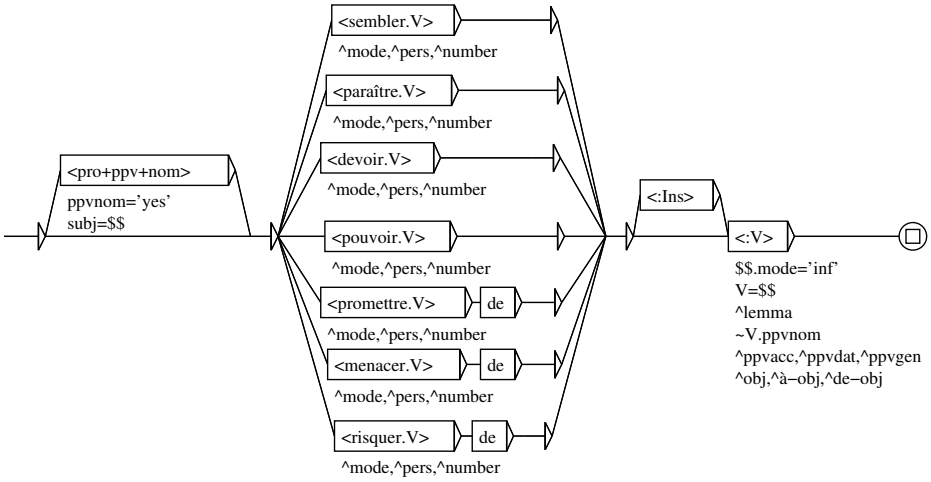

FIG. 4.46 – Sous-graphe `aux-mode`

ou aspectuels puisque certains de ces verbes (comme par exemple *hésiter à*, *oser*, *oublier de*, etc.) sélectionnent des sujets animés uniquement. Nous considérons ce type de verbes comme des prédicats à deux arguments (le sujet et la proposition infinitive), tels que le premier argument est nécessairement coréférent au sujet du second argument phrastique. En contrepartie, les verbes que nous avons sélectionnés dans notre grammaire n'imposent pas de restriction particulière sur la nature du sujet de la phrase, mais certaines restrictions peuvent néanmoins être imposées par le verbe principal :

*(il + *Luc) pleut*
*(il + *Luc) menace de pleuvoir*

En ce sens, nous les considérons comme des prédicats unaires à argument phrastique. A plus long terme, nous envisageons de refaire la grammaire des auxiliaires en la générant automatiquement à partir d'un sous-ensemble de la table 1 de laquelle nous aurions selectionné les entrées que nous aurions clairement identifiées comme des auxiliaires de mode ou d'aspect.

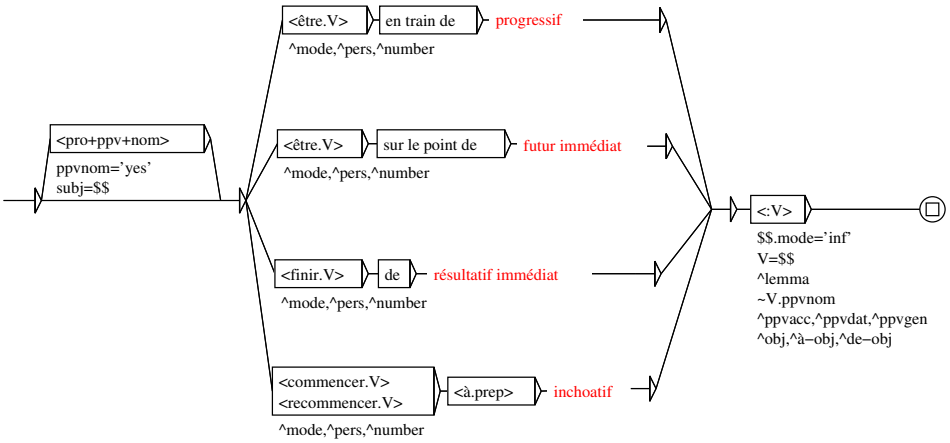

FIG. 4.47 – Sous-graphe `aux-aspect`

4.5.4 Exemples de tables

La construction de la grammaire complète du français est un long travail. Nous nous sommes pour l'instant concentré sur la syntaxe des verbes simples du français. Pour chacune de ces tables, nous n'avons pas nécessairement fait la description exhaustive de toutes les constructions. D'une façon générale, nous avons procédé de la manière suivante : nous faisons dans un premier temps, pour chaque table, la description des formes canoniques de ses entrées ; puis nous complétons notre grammaire au fur et à mesure que nous rencontrons de nouvelles constructions manquantes lors de son application pour l'analyse de texte. A l'heure actuelle nous avons ainsi entamé la conversion 18 tables de verbes (sur les 60 existantes). Nous avons également intégré la conversion de deux tables de noms prédicatif dans notre grammaire (les tables NDR1 et NAPE).

Nous montrons dans les pages qui suivent des extraits de notre meta-grammaire que nous pensons représentatifs des différents phénomènes de syntaxe que nous sommes parvenu à formaliser et analyser. Plus précisement, nous présentons les meta-grammaire de 3 tables : la table 36DT qui décrit les

verbes à deux compléments non phrastiques ; la table 12 des verbe à un complément phrastique et la table NDR1 de noms prédicatifs à verbes support *donner*.

Table 36DT

La table 36DT regroupe les verbes à trois arguments non phrastiques : un sujet, un complément d'objet direct et un complément datif humain introduit par la préposition *à*. Cette table contient également quelques verbes pour lesquels le second complément est introduit par la préposition *de* :

Paul a hérité cette pendule de son oncle

Afin de simplifier l'écriture de notre meta-grammaire, nous avons scindé la table en deux en fonction de la préposition introductrice. Nous présentons ici la meta-grammaire des verbes dont le second complément est introduit par *à*.

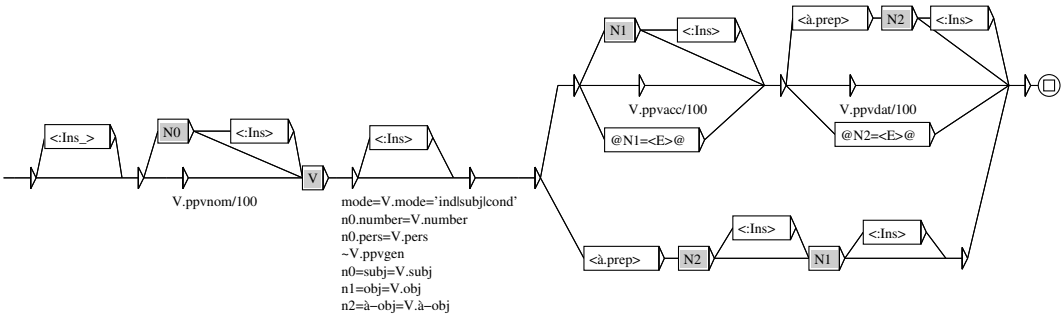

FIG. 4.48 – Formes de base pour la table 36DT

La figure 4.48 décrit les constructions canoniques des verbes de la table (de la forme : *N0 V N1 à N2*). Les équations sur les traits nous permettent entre autres :

– d'imposer que le verbe est conjugué à un temps fini (`mode='ind|subj|cond'`) ;

– de vérifier l'accord en nombre et en personne entre le verbe et son sujet (`n0.number=V.number` et `n0.pers=V.pers`) ;
– et d'identifier les arguments du verbe en fonction de leur position syntaxique (`n0=subj`, `n1=obj` et `n2=à-obj`).

Les arguments du verbe peuvent être réalisés syntaxiquement soit sous la forme d'un groupe nominal (décrit dans un sous-graphe de structuration dédié) soit par un pronom clitique, mais pas par les deux simultanément puisque dans ce cas la structure de traits ne pourrait pas s'unifier avec les deux valeurs qui sont incompatibles.

Nous avons isolé la description du verbe et de ses arguments dans des sous-graphes (présentés dans les figures 4.49 et 4.50) de manière à pouvoir les réutiliser facilement pour la description d'autres constructions.

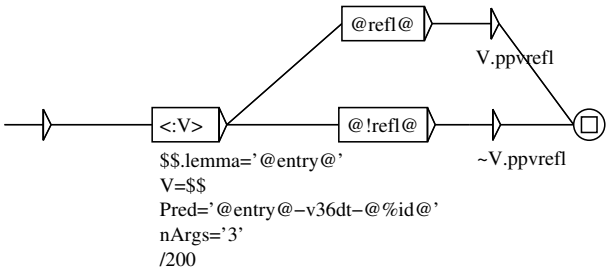

FIG. 4.49 – Sous-graphe `V`

Le sous-graphe `V` reconnaît simplement un non-terminal `V` comme décrit dans la section précédente. L'équation `$$.lemma='@entry@'` permet de vérifier que le lemme du noyau verbal est bien l'entrée de la table pour laquelle sera générée la grammaire lexicalisée. Les équations `Pred='@entry@-v36dt-@%id@'` et `nArgs='3'` servent à positionner les valeurs des traits `Pred` et `nArgs` selon les conventions que nous avons vues précédemment (cf. section 4.5.2). Enfin, la propriété `refl` est une propriété à valeur booléenne que nous avons ajoutée dans les tables que nous traitons et qui spécifie pour chaque entrée, si le verbe est employé avec un pronom réfléchi obligatoire pour cet emploi. Dans ce cas, l'équation `V.ppvrefl` permet de s'assurer que ce pronom est bien présent dans le noyau verbal reconnu (l'équation `~V.ppvrefl` permet de s'assurer de son absence dans le cas contraire).

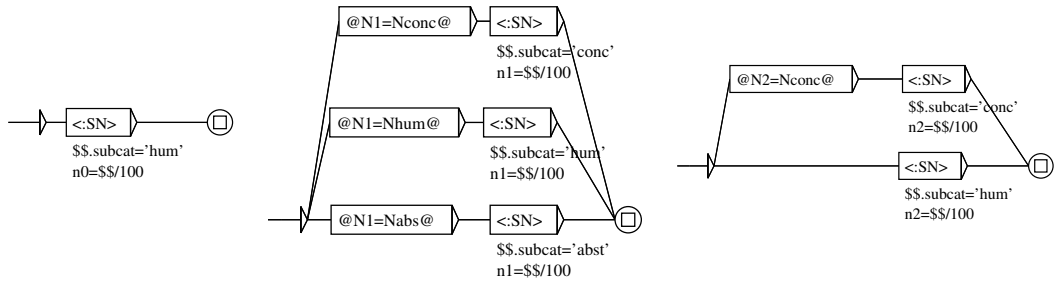

FIG. 4.50 – Sous-graphes `N0`, `N1` et `N2`

Les sous-graphes N0, N1 et N2 sont présentés dans la figure 4.50. Dans tous les cas, ces arguments sont réalisés sous la forme d'un SN. Les équations sur les traits permettent de préciser la catégorie sémantique du nom tête (nom humain, abstrait ou concret) en fonction des propriétés décrites dans la table. Nous avons pondéré chacune de ces descriptions par un poids de 100 ; en fait, nous avons pondéré ainsi dans toute notre grammaire, l'ensemble des chemins décrivant la réalisation d'un complément essentiel. Cette heuristique nous permet de réduire certaines ambiguïtés d'analyse en préférant les analyses dans lesquelles ont été identifiés le plus grand nombre de prédicats et d'arguments essentiels (par rapport aux analyses où le même complément aurait été identifié comme circonstanciel ou complément du nom par exemple).

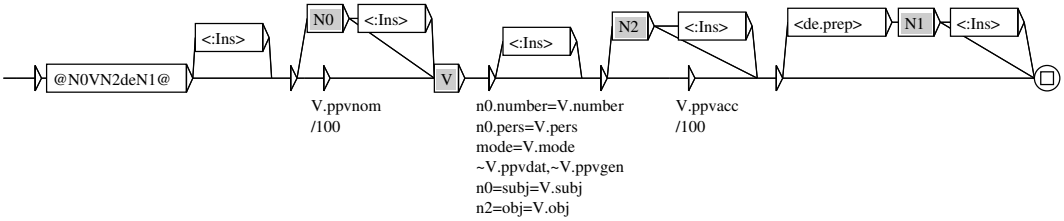

FIG. 4.51 – Sous-graphe `N0VN2deN1`

Le graphe de la figure 4.51 décrit les constructions associées à la propriété `N0VN2deN1` qui indique si le verbe admet la transformation de *croisement des arguments* avec laquelle l'argument N2 passe en position d'objet direct et l'argument N1 est introduit par la préposition *de* :

Max a arnaqué deux cent francs à Luc
= Max a arnaqué Luc de deux cent francs

Ainsi, par exemple, la séquence :

Luc a volé Lea

recoit deux analyses différentes avec notre grammaire :

```
voler-v36dt-271(Luc, Lea, _)
voler-v36dt-271(Luc, _, Lea)
```

Dans la première, correspondant à la structure de base, *Lea* est l'objet du vol (*Luc a volé Lea à ses parents*) ; dans la seconde, correspondant à la structure croisée, *Lea* est la personne qui a été volée (*Luc a volé Lea de 100 francs*). Cependant, la séquence suivante ne recevra qu'une seule analyse :

Luc a volé un œuf

En effet, le substantif *œuf* est un nom concret et ne peut donc pas être analysé comme l'argument N2 du verbe *voler* qui accepte uniquement un nom humain à cette position.

Enfin, nous présentons, dans la figure 4.52, le sous-graphe `P-prepN` décrivant les phrases avec un complément `miss` extrait et qui est appelé à partir du graphe général décrivant les propositions relatives par exemple (cf. figure 4.40). Nous distinguons deux possibilités : soit le complément extrait n'est pas prépositionnel (et dans ce cas le trait *missprep* a la valeur `zero`), nous reconnaissons alors un phrase privée d'un complément accusatif décrit dans le sous-graphe `P-Nacc` ; soit le complément manquant est un groupe nominal prépositionnel introduit par la préposition *à* (vérifié par l'équation `missprep='à'`), et dans ce cas, la phrase privée d'un tel complément est décrite dans le sous-graphe `P-aN`.

Nous présentons, pour exemple, le graphe `P-Nacc` dans la figure 4.53. Ce graphe reprend la structure du graphe décrivant les formes de base des verbes

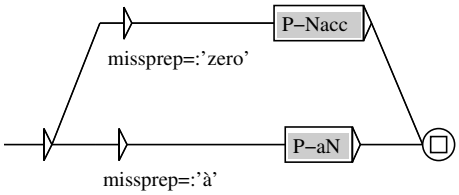

FIG. 4.52 – Sous-graphe `P-prepN`

de la table (cf. figure 4.48, seulement nous contraignons l'absence de l'argument N1 en surface qui ne peut être réalisé ni sous la forme d'un syntagme nominal (pas d'appel au sous-graphe `N1`) ni sous la forme d'un pronom clitique (équation `~V.ppvacc`). Nous identifions néanmoins cet argument comme le complément extrait avec l'équation `n1=obj=V.obj=miss`. Les équations sur les traits permettent de vérifier que la nature de cet argument manquant est bien compatible avec la nature de l'argument N1 attendu par la prédicat.

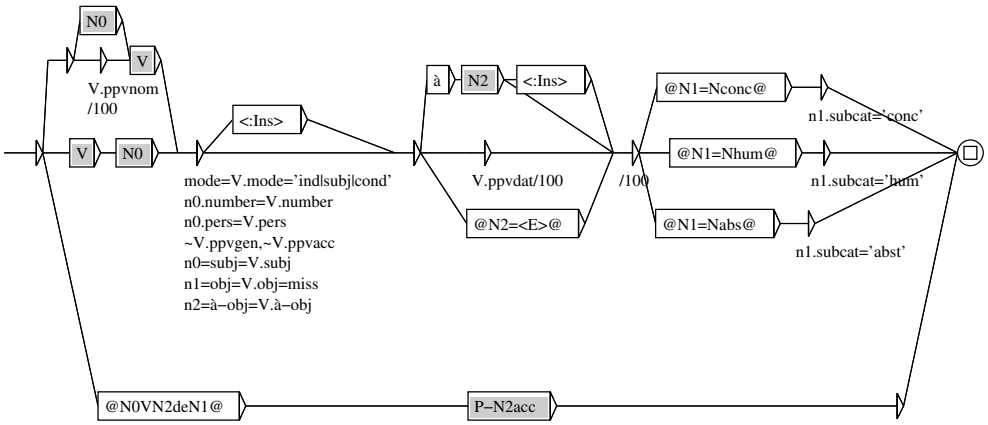

FIG. 4.53 – Sous-graphe P-Nacc

Notre grammaire analyse ainsi les SN suivants :

Lea que Luc a carotté de 100 francs.
Lea que Luc a remboursée

en leur associant les analyses :

```
rembourser-v36dt-215(Luc, _, Lea)
carotter-v36dt-37(Luc, francs, Lea)
```

Nous avons également décrit dans la meta-grammaire de la table les phrases participiales au présent (non terminal `P-ant`) et au passé (`Ppast`), les constructions infinitives (`Pinf`) ainsi que les infinitives privées d'un argument (`Pinf-prepN`). Le graphe `Pinf` est présenté pour exemple dans la figure 4.54. Sa forme reprend celle du graphe représentant les phrases canonique (cf. figure 4.48), excepté pour le sujet qui n'est pas réalisé syntaxiquement dans la proposition infinitive. Nous identifions néanmoins l'argument N0 du prédicat avec l'équation `n0=subj` ; comme nous le verrons dans la section suivante, la valeur de ce trait `subj` est positionnée depuis le graphe principal d'un verbe à complétive dans le cas où nous pouvons déterminer à quel complément du verbe principal est coréférent le sujet de l'infinitive.

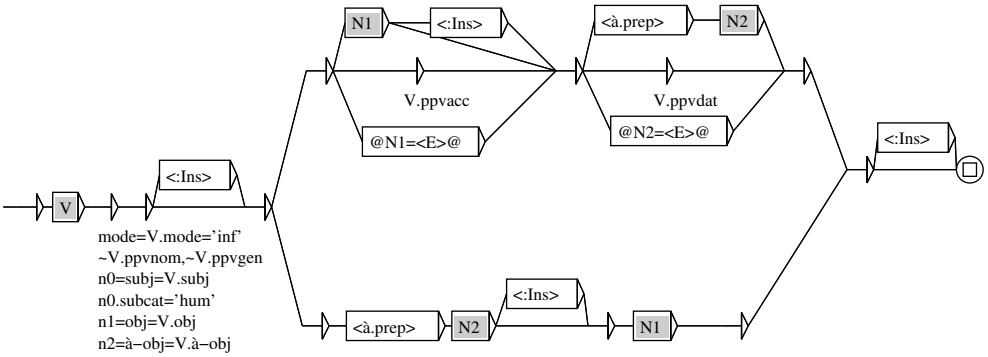

FIG. 4.54 – Sous-graphe `Pinf`

Table 12

La table 12 comporte les verbes à un complément phrastique qui ont la particularité (contrairement aux verbes de la table 6) de rentrer également dans une structure à deux compléments : le premier direct et le second sous

la forme d'une infinitive, introduite par la préposition *de*, dont le sujet est coréférent au premier complément. Par exemple :

Luc approuve que Max soit parti
= Luc approuve Max d'être parti

Nous décrivons dans le graphe de la figure 4.55 ces deux structures définitionnelles.

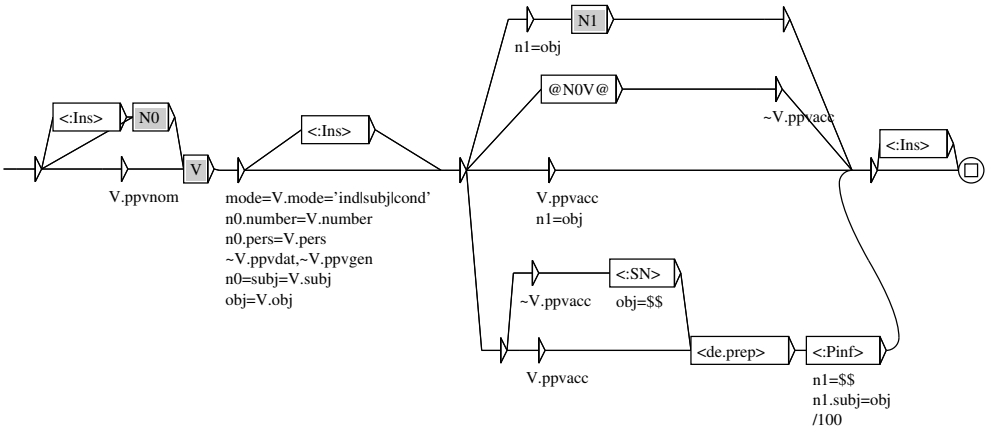

FIG. 4.55 – Formes de base (table 12)

Lorsque nous reconnaissons la construction à deux compléments, nous identifions la proposition infinitive comme l'argument N1 du prédicat (équation `n1=$$`) et nous rétablissons son sujet, coréférent au complément d'objet direct du verbe principal (équation `n1.subj=obj`). De cette manière, les deux séquences suivantes (considérées comme transformationnellement équivalentes dans le cadre du lexique-grammaire) :

Luc apprécie que Max ait donné une rose à Lea
Luc apprécie Max d'avoir donné une rose à Lea

recoivent exactement la même analyse en terme d'identification des prédicats et de leurs arguments :

apprécier-v12-9(Luc, donner-v36dt-81(Max, rose, Lea))

A notre connaissance, il n'existe pas d'autre analyseur syntaxique exploitant des ressources linguistiques à large couverture capable de mettre en relation ce type de constructions transformationnellement équivalentes.

Le sous-graphe `V` (figure 4.56) qui décrit le prédicat verbal est fait sous le même modèle que son homonyme de la table 36DT.

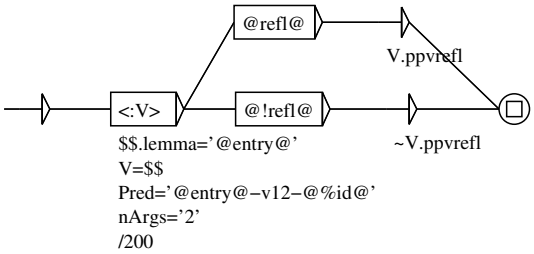

FIG. 4.56 – Sous-graphe `V`

Les figures 4.57 et 4.58 décrivent les différentes réalisations des arguments N0 et N1. Certains verbes de la table 12 acceptent un sujet non restreint (indiqué par la propriété `N0=Nnr`) :

(Max+qu'il fasse beau+la fatigue) empêche Luc de travailler

Les autres verbes acceptent un sujet humain uniquement :

*(Max+*qu'il fasse beau+*la fatigue) méprise Luc de travailler*

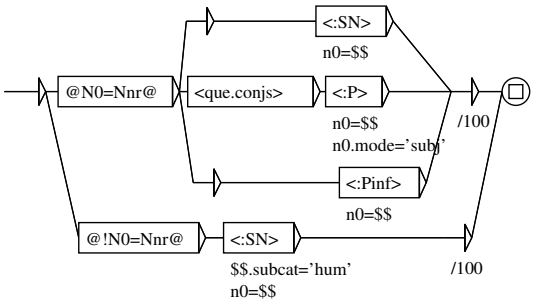

FIG. 4.57 – Sous-graphe `N0`

L'argument N1, quant à lui, peut être réalisé sous la forme d'une complétive à l'indicatif (propriété `N1=quePind`) ou au subjonctif (`N1=quePsubj`). Pour certains verbes, il est possible de mettre cette complétive à l'infinitif (introduite ou non par la préposition *de*), sans la présence d'un autre complément d'objet direct. Dans ce cas, le sujet de l'infinitive est coréférent au sujet du verbe principal (ce qui est formalisé avec l'équation `n1.subj=n0`) :

*Max adore (ε+*de) faire la vaisselle*
*Max arrête (*ε+de) faire la vaisselle*
Max exècre (ε+de) faire la vaisselle

Enfin, nous présentons dans la figure 4.59 la meta-grammaire de la table 12 décrivant le non terminal `P-prepN`. Nous distinguons deux cas :

- soit l'élément manquant est un argument du verbe ; Il s'agit alors d'un complément d'objet direct (ce qui est vérifié par l'équation contrainte `missprep= :'zero'`). Ce complément peut soit être l'argument N1 du prédicat (*La démission de l'ingénieur que Luc critique*) et dans ce cas l'équation `n1=miss` permet d'identifier l'argument extrait ; soit il s'agit du sujet de l'infinitive qui elle est réalisée syntaxiquement dans la construction (*L'ingénieur que Luc critique d'avoir démissioné*), dans ce cas, l'équation `n1.subj=miss` permet de rétablir cette relation.
- soit l'argument N1 est bien présent dans la construction mais sous la forme d'une proposition subordonnée dans laquelle il manque un élément (l'élément extrait) :

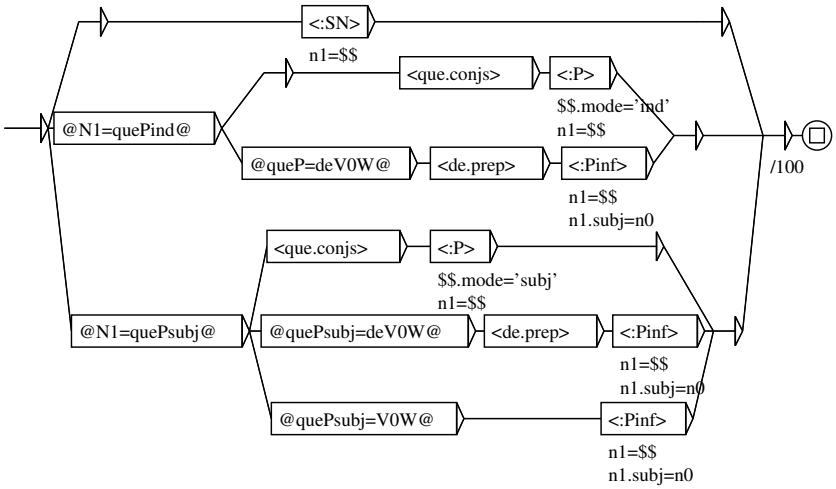

FIG. 4.58 – Sous-graphe `N1`

Max avec qui Lea apprécierait que Luc travaille
Max avec qui Lea apprécierait de travailler

Dans ces deux phrases, *Lea* est un argument du verbe *Max* et non du verbe *apprécier*. Les équations `^miss` et `^missprep` permettent de propager les informations sur l'élément extrait au niveau de la proposition enchâssée et ainsi de formaliser ce type de dépendances à distance.

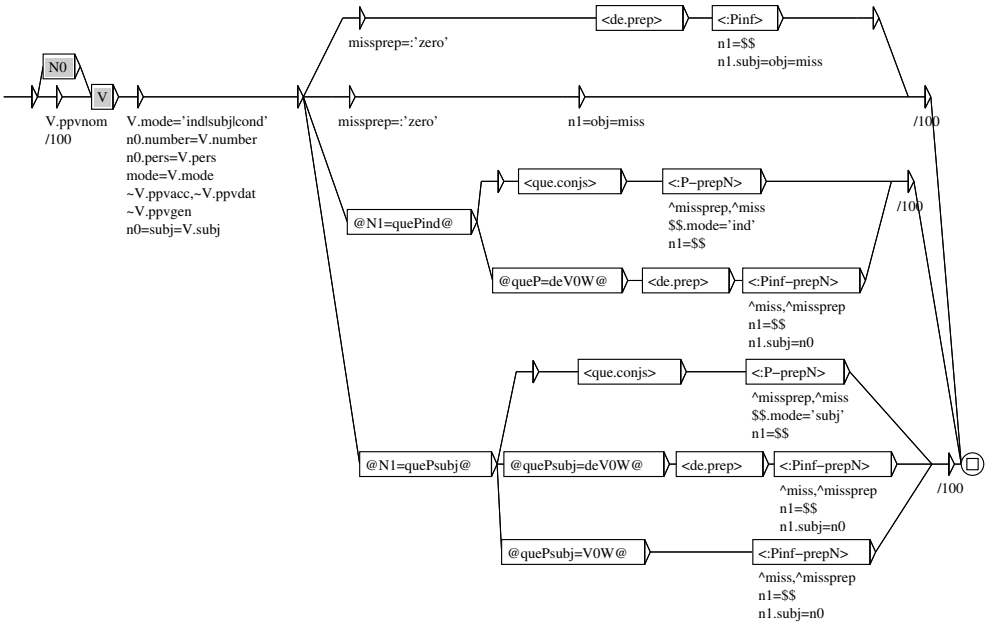

FIG. 4.59 – `P-prepN`

Table NDR1

Pour terminer cette présentation générale de notre grammaire, nous présentons nos travaux en cours concernant l'intégration des constructions à noms prédicatifs. A cet effet, nous avons entamé la conversion de la table NDR1 de Gaston Gross [Gross, 1989] décrivant les noms prédicatifs qui entrent dans une construction simple à verbe support *donner*, et qui acceptent également une construction converse à verbe support *recevoir*. Le substantif *gifle*, par exemple, appartient à cette table :

Luc a donné une gifle à Lea
= Lea a reçu une gifle de Luc

Nous nous sommes pour l'instant concentré sur la description du premier type de constructions (avec verbe support *donner*) ; le graphe principal décrivant ces formes canoniques est présenté dans la figure 4.60.

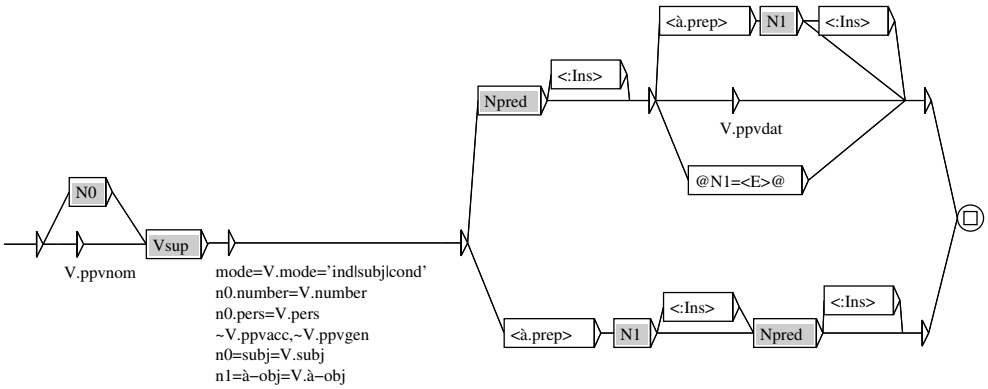

FIG. 4.60 – Constructions canoniques pour la table NDR1

Les graphes de la figure 4.61 présentent les sous-graphes `Vsup`, décrivant la réalisation du verbe support, et le sous-graphe `Npred`, décrivant le nom prédicatif. Nous avons décrit dans le graphe `Vsup`, les possibilités de variations lexicales au niveau du verbe support ; ces variations, qui ont été décrites dans la table, sont sujettes à des contraintes en fonction de l'entrée prédicative :

*Luc a (accordé+apporté+*flanqué) son aide à Lea*
*Luc a (*accordé+*apporté+flanqué) une punition à Lea*

Notre graphe `Npred` qui reconnaît le syntagme nominal à tête prédicative est pour l'instant très simple : il reconnaît simplement un non terminal SN dont le nom tête correspond à l'entrée de la table pour laquelle la grammaire sera lexicalisée (équation `$$.lemma='@entry@'`). Dans les faits, la forme des groupes nominaux dans lesquels apparaissent ces entrées prédicatives est soumise à certaines restrictions, notamment au niveau du déterminant :

*Luc donne (l'+*une) absolution à Lea*
*Luc donne (*le+un) baiser à Lea*

Ces restrictions étant décrites dans la table, nous pensons, à plus long terme, affiner notre grammaire en remplaçant l'appel au non terminal `SN` par un

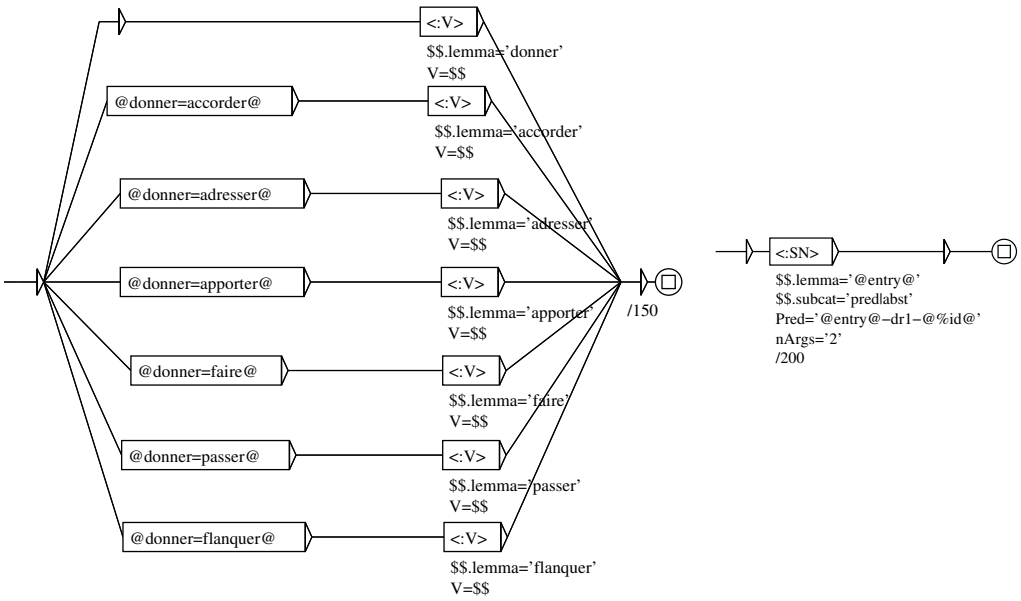

FIG. 4.61 – Sous-graphes `Vsup` et `Npred`

graphe lexicalisé tenant compte de ces propriétés.

Nous avons pondéré la reconnaissance du verbe support d'un poids de 150 et celle du nom prédicatif d'un poids de 200 (cf. figure 4.61). De cette manière, nous préférons, dans les cas d'ambiguïtés, les analyses correspondant aux constructions à verbe support et nom prédicatif (poids de 350) par rapport aux analyses identifiant la même séquence comme la combinaison d'un verbe plein accompagné d'un de ses compléments (d'un poids 200+100=300). Ainsi, si nous considérons la séquence suivante :

Luc donne une gifle à Lea

Cette séquence est reconnue par notre grammaire de deux façons différentes : dans la première analyse d'un poids de 500 (200 pour le prédicat + 300 pour les trois arguments), le verbe *donner* dans son emploi décrit dans la table 36DT est identifié comme le prédicat de la phrase et les substantifs *Luc*,

gifle et Lea comme ses trois arguments ; dans la seconde analyse, d'un poids de 550 (200 pour le prédicat + 150 pour le verbe support + 200 pour les deux arguments), le substantif *gifle* est le prédicat de la phrase et *Luc* et *Lea* sont ses deux arguments. La seconde analyse ayant un score supérieur, seule celle-ci subsiste à l'issue de l'analyse :

```
gifle-dr1-128(Luc, Lea)
```

Les noms prédicatifs n'apparaissent pas nécessairement accompagnés de leur verbe support dans les textes. Ils peuvent être à la tête d'un syntagme nominal dans lequel ses arguments peuvent éventuellement être réalisés sous la forme de compléments du nom :

Max s'étonne que Luc ait flanqué une gifle à Lea
= Max s'étonne de la gifle de Luc à Lea [nominalisation]

Ainsi, de manière à donner aux deux phrases précédentes la même analyse, nous avons commencé à compléter notre grammaire avec la description des SN complexes à noyau prédicatif. Le graphe de ce type de SN pour les entrées de la table NDR1 est présenté, sous sa forme péliminaire, dans la figure 4.62.

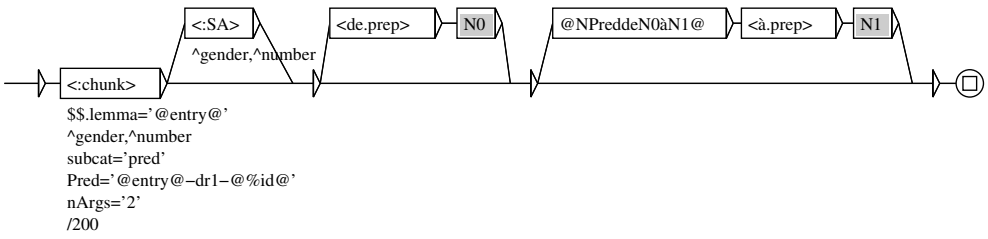

FIG. 4.62 – `SN` à tête prédicative

Nous décrivons dans ce graphe la possibilité de réaliser l'argument N0 du prédicat sous la forme d'un complément du nom introduit par la préposition *de*. L'argument N1 peut également apparaître au sein du même syntagme nominal, introduit par la préposition *à*, pour les entrées de la table qui acceptent ce type de constructions (propriété `NPreddeN0àN1`) :

Luc donne son soutien à Lea
= le soutien de Luc à Lea

Cependant, cette description est encore incomplète et même erronée pour certaines entrées. Pour certains noms, c'est l'argument N1 qui est introduit par la préposition *de*, l'argument N0 pouvant éventuellement être introduit par une autre préposition :

Luc accorde son émancipation à Lea
= l'émancipation de Lea (? par Luc)

Ces propriétés transformationnelles n'ayant pas été codées dans les tables, nous n'avons pas été en mesure de représenter de manière satisfaisante ce phénomène de nominalisation des phrases à verbe support. Il nous paraît donc nécessaire de compléter la description des tables des noms prédicatifs en étudiant et codant systématiquement ces propriétés manquantes de manière à pouvoir compléter et corriger notre grammaire.

4.6 Résultats

4.6.1 Evaluation sur le corpus TSNLP

Nous avons procédé à une évaluation de notre analyseur et de la couverture de notre grammaire dans son état actuel en l'appliquant sur le corpus TSNLP pour le français [Balkan *et al.*, 1994]. Ce corpus consiste en un jeu de phrases-test représentatif des différentes constructions syntaxiques du français. Nous avons manuellement supprimé de ce corpus les phrases interrogatives et impératives car ces constructions ne sont pas pour l'instant décrites dans notre grammaire. Le corpus résultant est composé de 1 168 phrases. Ce texte a été analysé par notre analyseur en 8 minutes et 20 secondes, ce qui donne une moyenne de 0,42 seconde par phrase. Au total 778 phrases ont reçu une analyse correcte. Nous obtenons donc une couverture de 66% pour l'analyse de ce texte.

La majorité des erreurs d'analyse sont dues à la présence de syntagmes nominaux et adjectivaux complexes à tête prédicative dont le cadre de sous-catégorisation n'est pas décrit dans notre grammaire, comme dans les phrases :

L'ingénieur a obtenu l'accord de l'entreprise pour venir
Jean est affectueux avec l'ingénieur
Jean est heureux que l'ingénieur vienne
Jean est jaloux de l'ingénieur

Toutes ces erreurs n'invalident pas notre méthode : il suffit de décrire le comportement des items lexicaux manquants afin d'élargir la couverture lexicale de notre grammaire et d'analyser ces constructions. Il en est de même pour la description des expressions figées ou locutions verbales, dont on trouve certaines occurrences dans notre corpus de test :

Si l'ingénieur était venu, la réunion aurait eu lieu.

En revanche, d'autres erreurs sont dues à des constructions syntaxiques non décrites dans notre grammaire. La plupart d'entre elles mettent en jeu l'utilisation complexe de conjonctions :

L'ingénieur obtient l'accord et l'homme la permission de l'entreprise pour venir.
L'ingénieur a accepté et signé le contrat

Dans la première phrase, la conjonction *et* coordonne deux phrases qui ont le même verbe en commun (ici, un verbe support) qui est réalisé uniquement dans la première proposition. Pour analyser cette séquence, il faudrait donc mettre en place un mécanisme permettant de rétablir le verbe manquant dans la seconde proposition. Un tel mécanisme peut sans doute être réalisé en propageant les informations sur l'élément manquant dans la seconde proposition à l'aide de contraintes d'unification mais risque d'avoir un impact négatif sur l'efficacité de notre analyseur. La seconde phrase est sans doute encore plus complexe à analyser correctement puisque l'auxiliaire *avoir* modifie les deux verbes au participe passé. De plus, les noms *ingénieur* et *contrat*

sont simultanément arguments des deux prédicats *accepter* et *signer*. Nous n'avons pas pour l'instant trouvé de solution simple pour analyser ce genre de constructions. A notre connaissance, les grammaires formelles du français les plus reconnues pour leur large couverture (telles que FTAG [Abeillé, 2002] ou LFG [Boullier et Sagot, 2005]) ne traitent pas ce type de constructions complexes de manière satisfaisante.

En revanche, nous pensons que notre travail a contribué à faire avancer la recherche en analyse syntaxique sur les deux points suivants, généralement peu ou pas traités dans les différentes approches courantes :

Analyse syntaxique profonde

Beaucoup des analyseurs syntaxiques existants produisent comme résultat un arbre de dérivation représentant la structure syntagmatique de l'énoncé traité (par exemple, la campagne récente EASY d'évaluation des analyseurs syntaxiques du français, demandait aux candidats de produire ce type de résultat). Dans le cadre de notre travail, nous avons tenté de faire de l'analyse syntaxique plus profonde : outre la production d'un arbre de dérivation, nous nous sommes essentiellement intéressé à l'identification des prédicats syntaxiques et de leurs arguments, et ce, indépendamment de leur position ou fonction syntaxique dans l'énoncé. Ce point nous permet notamment de mettre en relation des constructions considérées comme transformationnellement équivalentes, comme les séquences suivantes :

Max apprécie que Luc ait donné une gifle à Lea
Max apprécie la gifle que Luc a donnée à Lea
Max apprécie la gifle de Luc à lea
la gifle de Luc à lea est appréciée (par+de) Max

qui reçoivent exactement la même analyse (en terme de relations prédicat-argument), à l'issue de l'analyse par notre grammaire :

```
apprécier-v12-9(Max, gifle-dr1-128(Luc, Lea))
```

Distinction des emplois

A l'inverse, un même lemme peut avoir plusieurs sens différents en fonction de la phrase dans laquelle il apparaît. Par exemple, au moins quatre emplois distincts ont été repertoriés pour le verbe *voler* dans le cadre des travaux effectués au LADL :

Max vole chercher Lea (table 2)
Max a volé dans ce livre que Napoleon est mort (table 10)
L'avion vole vers Paris (table 35L)
Max a volé une montre à Lea (table 36DT)

Les distinctions entre ces emplois ayant été faites sur des critères purement formels, nous avons pu les formaliser dans notre grammaire. Nous obtenons donc, à l'issue de l'analyse de ces séquences, des résultats dans lesquels ces emplois ont correctement été distingués :

```
voler-v2-107(Max, _, chercher-v32r2-87(Max, Lea))
voler-v10-194(Max, mort(Napoleon), dans(livre))
voler-v35l-280(avion, vers(Paris))
voler-v36dt-271(Max, montre, Lea)
```

Chapitre 5

Conclusion

Nos recherches ont porté sur le traitement automatique de textes par application de grammaires lexicalisées en utilisant des ressources linguistiques à large couverture. Dans ce contexte, nous avons effectué nos recherches dans trois domaines : l'algorithmie, la création d'applications utiles dans un contexte industriel, et l'analyse syntaxique profonde.

Au niveau algorithmique, nous avons précisé le modèle formel des grammaires locales et nous proposons des algorithmes d'optimisation de ce type de grammaire en préalable à leurs utilisations pour l'analyse. De même, nous proposons un algorithme efficace pour l'application de ces grammaires sur un texte. Notre algorithme améliore le traitement des différents types d'ambiguïtés : les ambiguïtés lexicales (en représentant le texte sous la forme d'un automate acyclique) et les ambiguïtés syntaxiques (en représentant les résultats d'analyses sous la forme compacte d'une forêt partagée d'arbres de dérivation décorés). Nous avons montré par des évaluations chiffrées, que nos algorithmes permettent de traiter de gros volumes de données textuelles (corpus de plusieurs millions de mots) en combinaison avec des ressources linguistiques fines et à large couverture (dictionnaires électroniques de plus d'un million d'entrées et grammaire de plusieurs milliers d'états).

Au niveau applicatif, nous avons participé au développement de la plateforme logicielle Outilex, qui implémente les opérations de base pour le traite-

ment de textes écrits (traitements sans ressources linguistiques, traitements avec lexiques et traitement par grammaires). L'architecture modulaire de la plate-forme et sa licence peu restrictive (LGPL) devraient permettre la création, à faible coût, d'applications de plus haut niveau par les acteurs universitaires et industriels du domaine. De plus, l'intégration de la notion de pondération dans les grammaires ouvre la voie à la réalisation de nouvelles applications hybrides, mélangeant méthodes symboliques à base de ressources linguistiques avec et méthodes stochastiques.

Enfin, dans le cadre de nos recherches en analyse syntaxique, nous avons poursuivi le travail de Sébastien Paumier [Paumier, 2003a] consistant à exploiter les tables du lexique-grammaire pour faire de l'analyse fine et identifier les phrases simples (les prédicats et leurs arguments) dans les énoncés complexes. A cet effet, nous avons fait évoluer le formalisme grammatical vers un formalisme à structure de traits, en augmentant les grammaires WRTN d'une composante d'unification. Les équations sur les traits qui décorent notre grammaire nous permettent de résoudre de manière naturelle et déclarative différents phénomènes syntaxiques et aussi de représenter de manière formelle les résultats de l'analyse.

Les perspectives de nos travaux sont multiples : dans le cadre de nos développements de la plate-forme Outilex, nous avons mis en place les briques de base permettant la réalisation à faible coût de nouvelles applications dédiées à de nombreux types de traitements automatiques sur les textes écrits. De plus, nous avons conçu le formalisme original des grammaires WRTN pour faciliter la mise en place de traitements combinant les méthodes statistiques avec les méthodes à base de ressources linguistiques, mais nous n'avons pas exploité réellement cette fonctionnalité dans nos travaux jusqu'à présent. Ainsi, nous souhaiterions, par la suite, approfondir cette voie en réalisant des applications de plus haut niveau et utilisables dans un contexte industriel, notamment dans le domaine de l'extraction d'information, domaine pour lequel nous pensons que le formalisme WRTN est particulièrement adapté.

Enfin, à l'heure actuelle, la couverture de notre grammaire lexicalisée du français est encore faible tant au niveau lexical que syntaxique. Nous souhaitons ainsi poursuivre nos travaux dans ce sens en l'enrichissant de nouvelles constructions et de nouveaux éléments prédicatifs. Cependant, la réalisation de grammaires complètes et linguistiquement correctes s'inscrit sur le long

terme et demandera nécessairement la collaboration de plusieurs chercheurs notamment en linguistique. De plus, la description de l'ensemble des éléments prédicatifs n'est pas achevée, notamment en ce qui concerne les noms et les adjectifs ; la complétion de cette base de données linguistique demandera encore sûrement de nombreuses années de travail. Nous espérons que la possibilité d'évaluer les données existantes en les exploitant dans un contexte d'analyse syntaxique sera une stimulation supplémentaire pour la maintenance et l'enrichissement des tables du lexique-grammaire.

Bibliographie

[Abeillé, 1991] Abeillé, A. (1991). *Une grammaire lexicalisée d'arbres adjoints pour le français*. Thèse de doctorat, Université Paris 7.

[Abeillé, 1993] Abeillé, A. (1993). *Les nouvelles syntaxes : grammaires d'unification et analyse du français*. Armand Colin, Paris.

[Abeillé, 2002] Abeillé, A. (2002). *Une grammaire électronique du français*. CNRS Editions, Paris.

[Abeillé et Blache, 2000] Abeillé, A. et Blache, P. (2000). Grammaires et analyseurs syntaxiques. *In* Perriel, J.-M., éditeur : *Ingénierie des Langues*. Hermès, Paris.

[Abney, 1996] Abney, S. (1996). Partial parsing via finite-state cascades. *In Workshop on Robust Parsing, 8th European Summer School in Logic, Language and Information*, pages 8–15, Prague, Czech Republic.

[Auber, 2003] Auber, D. (2003). Tulip : A huge graph visualisation framework. *In* Mutzel, P. et Jünger, M., éditeurs : *Graph Drawing Software (Mathematics and Visualization)*, pages 105–126. Springer-Verlag.

[Balkan *et al.*, 1994] Balkan, L., Meijer, S., Arnold, D., Estival, D., Falkedal, K., Lehrmann, S., Dauphin, E., Regnier-Prost, S., Netter, K. et Oepen, S. (1994). Test suite design annotation scheme. Rapport technique D-WP2.2, Essex.

[Barrier, 2006] Barrier, S. (2006). *Une métagrammaire pour les noms prédicatifs du français. Développement et experimentations pour les grammaires TAG*. Thèse de doctorat, Université Paris 7.

[Bellman, 1957] Bellman, R. E. (1957). *Dynamic programming*. Princeton University Press, Princeton.

[Berstel, 1979] Berstel, J. (1979). *Transductions and Context-Free Languages*. Teubner Studienbücher, Stuttgart.

[Bird et Loper, 2004] Bird, S. et Loper, E. (2004). NLTK : the natural language toolkit. *In Proc. of ACL*.

[Blanc et Constant, 2005] Blanc, O. et Constant, M. (2005). Lexicalization of grammars with parameterized graphs. *In Proc. of RANLP 2005*, pages 117–121, Borovets, Bulgarie. INCOMA Ltd.

[Blanc et Constant, 2006] Blanc, O. et Constant, M. (2006). Outilex, a linguistic platform for text processing. *In Proc. of ACL 2006*.

[Blanc et Dister, 2004] Blanc, O. et Dister, A. (2004). Automates lexicaux avec structure de traits. *In Actes de RECITAL*, pages 23–32.

[Blanc *et al.*, 2005] Blanc, O., Ioannidou, K. et Voskaki, R. (2005). Automatic elimination of lexical ambiguities in modern greek : presentation of the elag system. *In Studies in Greek Linguistics, Proceedings of the 25th Annual Meeting of the Department of Linguistics, Faculty of Philosophy, Aristotle University of Thessaloniki*, pages 89–100.

[Boons *et al.*, 1976a] Boons, J.-P., Guillet, A. et Leclère, C. (1976a). La structure des phrases simples en français : classes de constructions transitives. Rapport technique, LADL, Université Paris 7.

[Boons *et al.*, 1976b] Boons, J.-P., Guillet, A. et Leclère, C. (1976b). *La structure des phrases simples en français : constructions intransitives*. Droz, Genève.

[Boullier et Sagot, 2005] Boullier, P. et Sagot, B. (2005). Analyse syntaxique profonde à grande échelle : Sxlfg. *Traitement Automatique des Langues*, 46(2).

[Bresnan, 1982] Bresnan, J., éditeur (1982). *The Mental Representation of Grammatical Relations*. MIT Press, Cambridge, MA.

[Brill, 1995] Brill, E. (1995). Transformation-based error-driven learning and natural language processing : A case study in part-of-speech tagging. *Computational Linguistics*, 21(4):543–565.

[Camugli Gallardo et Blanc, 2006] Camugli Gallardo, C. et Blanc, O. (2006). Le fonctionnement injonctif d'expressions figées en italien. *In* (ed), M. H. A. C., éditeur : *Venez, venez ! De la suggestion à l'injonction dans les langues romanes. Travaux et documents*.

[Choffrut, 1977] CHOFFRUT, C. (1977). Une caracterisation des fonctions sequentielles et des fonctions sous-sequentielles en tant que relations rationnelles. *Theor. Comput. Sci.*, 5(3):325–337.

[Choffrut, 1978] CHOFFRUT, C. (1978). *Contributions à l'étude de quelques familles remarquables de fonctions rationnelles*. Thèse de doctorat, Université de Paris 7.

[Chomsky, 1965] CHOMSKY, N. (1965). *Aspects of the theory of syntax*. MIT Press, Cambridge.

[Chrobot, 2000] CHROBOT, A. (2000). Description des déterminants numéraux anglais par automates et transducteurs finis.

[Church, 1988] CHURCH, K. W. (1988). A stochastic parts program and noun phrase parser for unrestricted text. *In Proceedings of the second conference on Applied natural language processing*, pages 136–143, Morristown, NJ, USA. Association for Computational Linguistics.

[Clément et de la Clergerie, 2005] CLÉMENT, L. et de la CLERGERIE, É. (2005). MAF : a morphosyntactic annotation framework. *In Proc. of the Language and Technology Conference, Poznan, Poland*, pages 90–94.

[Constant, 2000] CONSTANT, M. (2000). Description d'expressions numériques en français.

[Courtois, 1990] COURTOIS, B. (1990). Un système de dictionnaires électroniques pour les mots simples du français. *Langue Française*, 87:11–22.

[Courtois, 2004] COURTOIS, B. (2004). Dictionnaires électroniques DELAF anglais et français. *In* LECLÈRE, C., LAPORTE, É., PIOT, M. et SILBERZTEIN, M., éditeurs : *Lexique, Syntaxe et Lexique-Grammaire/Syntax, Lexis and Lexicon-Grammar*, Lingvisticae Investigationes Supplementa, pages 113–123. Benjamins, Amsterdam/Philadelphie.

[Cunningham, 2002] CUNNINGHAM, H. (2002). GATE, a general architecture for text engineering. *Computers and the Humanities*, 36:223–254.

[Cunqueiro, 2005] CUNQUEIRO, C. B. (2005). Lösung von lexikalischen ambiguitäten in der spanischen sprache mittels des formalismus elag.

[Danlos, 2005a] DANLOS, L. (2005a). Automatic recognition of French expletive pronoun occurrences. *In Companion Volume of the International Joint Conference on Natural Language Processing, Jeju, Korea*, page 2013.

[Danlos, 2005b] DANLOS, L. (2005b). Ilimp : Outil pour repérer les occurences du pronom impersonnel il. *In* JARDINO, M., éditeur : *Actes de*

TALN 2005 (Traitement automatique des langues naturelles), Dourdan. ATALA, LIMSI.

[de Negroni-Peyre, 1978] de Negroni-Peyre, D. (1978). Nominalisations par être en et réflexivation (admiration, opposition, révolte et rage). *Linguisticae Investigationes*, 2(1):127–163.

[EAGLES, 1996] EAGLES (1996). *EAGLES Annotation Guidelines*.

[Earley, 1970] Earley, J. (1970). An efficient context-free parsing algorithm. *Comm. ACM*, 13(2):94–102.

[Fairon, 2000] Fairon, C. (2000). *Structures non connexes : grammaire des incises en français : description linguistique et outils informatiques*. Thèse de doctorat, Université Paris 7.

[Francopoulo, 2003] Francopoulo, G. (2003). *Proposition de norme des lexiques pour le traitement automatique du langage*. AFNOR. 21 p.

[Friburger, 2002] Friburger, N. (2002). *Reconnaissance automatique des noms propres : application à la classification automatique de textes journalistiques*. Thèse de doctorat, Université de Tours.

[Gansner et North, 2000] Gansner, E. R. et North, S. C. (2000). An open graph visualization system and its applications to software engineering. *Software — Practice and Experience*, 30(11):1203–1233.

[Gardent *et al.*, 2005] Gardent, C., Guillaume, B., Perrier, G. et Falk, I. (2005). Maurice gross' grammar lexicon and natural language processing. *In Proceedings of the 2nd Language and Technology Conference*.

[Gazdar *et al.*, 1985] Gazdar, G., Klein, E., Pullum, G. K. et Sag, I. (1985). *Generalized Phrase Structure Grammar*. Harvard University Press. Cambridge Massachusetts.

[George, 2003] George, M. (2003). *Terminology and other language resources. Lexical Resource Markup Framework*. ISO. 16 p.

[Giry-Schneider, 1978] Giry-Schneider, J. (1978). *Les nominalisations en français. L'opérateur faire dans le lexique*. Droz, Genève.

[Goscinny et Sempé, 2006] Goscinny et Sempé (2006). *Histoires inédites du Petit Nicolas*, volume 2. IMAV Editions.

[Greibach, 1965] Greibach, S. A. (1965). A new normal-form theorem for context-free phrase structure grammars. *J. ACM*, 12(1):42–52.

[Gross, 1989] Gross, G. (1989). *Etude syntaxique de constructions converses*. Droz, Genève-Paris.

[Gross, 1975] Gross, M. (1975). *Méthodes en syntaxe*. Hermann, Paris.

[Gross, 1993] Gross, M. (1993). Local grammars and their representation by finite automata. *In* Hoey, M., éditeur : *Data, Description, Discourse, Papers on the English Language in honour of John McH Sinclair*, pages 26–38. Harper-Collins, London.

[Gross, 1997] Gross, M. (1997). The construction of local grammars. *In* Roche, E. et Schabès, Y., éditeurs : *Finite-state language processing*, Language, Speech, and Communication Series, pages 329–354. MIT Press, Cambridge (Mass.).

[Gross, 2001] Gross, M. (2001). Grammaires locales de déterminants nominaux. *LIS*, 23:221–246.

[Gross et Senellart, 1998] Gross, M. et Senellart, J. (1998). Nouvelles bases statistiques pour les mots du français. *In Actes des 4èmes Journées Internationnales d'Analyse des Données Textuelles*.

[Guillet et Leclère, 1992] Guillet, A. et Leclère, C. (1992). *La structure des phrases simples en français 2 : les constructions transitives locatives*. Droz, Genève.

[Harris, 1951] Harris, Z. S. (1951). *Structural Linguistics*. The University of Chicago Press, 1960, Chicago and London.

[Harris, 1968] Harris, Z. S. (1968). *Mathematical Structures of Language*. Wiley Interscience, New York.

[Hopcroft et Ullman, 1979] Hopcroft, J. et Ullman, J. (1979). *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, Massachusetts.

[Joshi et Hopely, 1996] Joshi, A. K. et Hopely, P. (1996). A parser from antiquity. *Nat. Lang. Eng.*, 2(4):291–294.

[Joshi *et al.*, 1975] Joshi, A. K., Levy, L. S. et Takahashi, M. (1975). Tree adjunct grammars. *J. Comput. Syst. Sci.*, 10(1):136–163.

[Jurafsky et Martin, 2000] Jurafsky, D. et Martin, J. (2000). *Speech and language processing*. Prentice Hall. 934 p.

[Kaplan et Kay, 1994] Kaplan, R. M. et Kay, M. (1994). Regular models of phonological rule systems. *Comput. Linguist.*, 20(3):331–378.

[Karttunen *et al.*, 1997] Karttunen, L., Gaál, T. et Kempe, A. (1997). Xerox finite-state tool. Rapport technique, Centre de recherche Xerox de Grenoble.

[Karttunen *et al.*, 1992] Karttunen, L., Kaplan, R. M. et Zaenen, A. (1992). Two-level morphology with composition. *In Proceedings of the 14th conference on Computational linguistics*, pages 141–148, Morristown, NJ, USA. Association for Computational Linguistics.

[Koskenniemi, 1990] Koskenniemi, K. (1990). Finite-state parsing and disambiguation. *In Proceedings of the 13th conference on Computational linguistics*, pages 229–232, Morristown, NJ, USA. Association for Computational Linguistics.

[Krstev *et al.*, 2005] Krstev, C., Vitas, D., Maurel, D. et Tran, M. (2005). Multilingual ontology of proper names. *In Proc. of the Language and Technology Conference, Poznan, Poland*, pages 116–119.

[Labelle, 1974] Labelle, J. (1974). *Etude de constructions avec opérateur avoir (nominalisations et extensions).* Thèse de doctorat, LADL, Université Paris 7.

[Laporte, 2005] Laporte, É. (2005). Lexicon management and standard formats. *In Proc. of the Language and Technology Conference, Poznan, Poland*, pages 318–322.

[Laporte et Monceaux, 1999] Laporte, É. et Monceaux, A. (1999). Elimination of lexical ambiguities by grammars : The elag system. pages 341—-367.

[Laporte, 1997] Laporte, Éric. (1997). Rational transductions for phonetic conversion and phonology. *In* Roche, E. et Schabès, Y., éditeurs : *Finite-state language processing*, Language, Speech, and Communication Series, chapitre 14, pages 407–428. MIT Press, Cambridge (Mass.).

[Leclère, 1990] Leclère, C. (1990). Organisation du lexique-grammaire des verbes français. *Langue Française.*

[Loper et Bird, 2002] Loper, E. et Bird, S. (2002). NLTK : the natural language toolkit. *In Proc. of the ACL Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics, Philadelphia.*

[Mason, 2004] Mason, O. (2004). Automatic processing of local grammar patterns. *In Proc. of the 7th Annual CLUK (the UK special-interest group for computational linguistics) Research Colloquium.*

[Maurel, 1990] Maurel, D. (1990). Adverbes de date : étude préliminaire à leur traitement automatique. *Lingvisticae Investigationes*, 14(1):31–63.

[Meunier, 1981] Meunier, A. (1981). *Nominalisations d'adjectifs par verbes supports*. Thèse de doctorat, LADL, Université Paris 7.

[Mohri, 1994] Mohri, M. (1994). On some applications of finite-state automata theory to natural language processing : Representation of morphological dictionaries, compaction, and indexation. Technical Report IGM 94-22, Institut Gaspard-Monge, Université de Marne-la-Vallée.

[Mohri, 1996] Mohri, M. (1996). On some applications of finite-state automata theory to natural language processing. *Natural Language Engineering*, 2(01):61–80.

[Mohri, 1997] Mohri, M. (1997). Finite-state transducers in language and speech processing. *Computational Linguistics*, 23(2):269–311.

[Mohri *et al.*, 1998] Mohri, M., Pereira, F. et Riley, M. (1998). A rational design for a weighted finite-state transducer library. *Lecture Notes in Computer Science*, 1436.

[Nakamura, 2005] Nakamura, T. (2005). Analysing texts in a specific domain with local grammars : The case of stock exchange market reports. *In Linguistic Informatics - State of the Art and the Future*, pages 76–98. Benjamins, Amsterdam/Philadelphia.

[Nasr, 2004] Nasr, A. (2004). Analyse syntaxique probabiliste pour grammaires de dépendances extraites automatiquement.

[Normier et Nossin, 1990] Normier, B. et Nossin, M. (1990). Genelex project : Eureka for linguistic engineering. *In Proc. of the International Workshop on Electronic Dictionaries, OISA, Kanagawa, Japan*, pages 63–70.

[Ozdowska et Claveau, 2005] Ozdowska, S. et Claveau, V. (2005). Alignement de mots par apprentissage de règles de propagation syntaxique en corpus de taille restreinte. *In* Jardino, M., éditeur : *Actes de TALN 2005 (Traitement automatique des langues naturelles)*, pages 243–252, Dourdan. ATALA, LIMSI.

[Paumier, 2000] Paumier, S. (2000). Nouvelles méthodes pour la recherche d'expressions dans de grands corpus. pages 289–295.

[Paumier, 2001] Paumier, S. (2001). Some remarks on the application of a lexicon-grammar. *Lingvisticae Investigationes*, 24(2):245–256.

[Paumier, 2003a] PAUMIER, S. (2003a). *De la reconnaissance de formes linguistiques à l'analyse syntaxique. Volume 1.* Thèse de doctorat, IGM, Université de Marne-la-Vallée.

[Paumier, 2003b] PAUMIER, S. (2003b). *De la reconnaissance de formes linguistiques à l'analyse syntaxique. Volume 2, Manuel d'Unitex.* Thèse de doctorat, IGM, Université de Marne-la-Vallée.

[Paumier, 2003c] PAUMIER, S. (2003c). A time-efficient token representation for parsers. *In Proc. of the EACL Workshop on Finite-State Methods in Natural Language Processing, Budapest*, pages 83–90.

[Pereira et Warren, 1980] PEREIRA, F. C. N. et WARREN, D. H. D. (1980). Definite clause grammars for language analysis - a survey of the formalism and a comparison with augmented transition networks. *Artif. Intell.*, 13(3): 231–278.

[Pierrel, 2000] PIERREL, J.-M. (2000). *Ingénierie des Langues.* Editions Hermès. 354 pages.

[Poibeau, 2001] POIBEAU, T. (2001). Extraction d'information dans les bases de données textuelles en génomique au moyen de transducteurs à états finis. *In* MAUREL, D., éditeur : *Actes de TALN 2001 (Traitement automatique des langues naturelles)*, pages 295–304, Tours. ATALA, Université de Tours.

[Poirier, 1999] POIRIER, H. (1999). *The XELDA framework.* http ://www.dcs.shef.ac.uk/ hamish/dalr/baslow/xelda.pdf.

[Pollard et Sag, 1994] POLLARD, C. et SAG, I. A. (1994). *Head-Driven Phrase Structure Grammar.* University of Chicago Press and CSLI Publications, Chicago, Illinois.

[Rajman, 1995] RAJMAN, M. (1995). Approche probabiliste de l'analyse syntaxique. *TAL*, 36-1-2:157–200.

[Revuz, 1991] REVUZ, D. (1991). *Dictionnaires et lexiques : méthodes et algorithmes.* Thèse de doctorat, Université Paris 7.

[Roche, 1993] ROCHE, E. (1993). *Analyse syntaxique transformationnelle du français par transducteurs et lexique-grammaire.* Thèse de doctorat, Université Paris 7.

[Roche et Schabes, 1997] ROCHE, E. et SCHABES, Y., éditeurs (1997). *Finite-State Language Processing.* MIT Press, Cambridge, MA, USA.

[Role, 1999] ROLE, F. (1999). La DTD TEI. *In Techniques de l'ingénieur - traité informatique*, volume H7 158.

[Sastre, 2005] SASTRE, J. M. (2005). XML-based representation formats of local grammars for NLP. *In Proc. of the Language and Technology Conference, Poznan, Poland*, pages 314–317.

[Schmid, 1994] SCHMID, H. (1994). Probabilistic part-of-speech tagging using decision trees. *Proceedings of the International Conference on New Methods in Language Processing*.

[Senellart, 1998] SENELLART, J. (1998). Reconnaissance automatique des entrées du lexique-grammaire des phrases figées. *Travaux de linguistique*, 37:109–125.

[Senellart, 1999] SENELLART, J. (1999). *Outils de reconnaissance d'expressions linguistiques complexes dans des grands corpus*. Thèse de doctorat, Université Paris 7.

[Shieber, 1986] SHIEBER, S. (1986). *An introduction to unification-based theories of grammar*. CSLI, University of Chicago Press.

[Silberztein, 1993] SILBERZTEIN, M. (1993). *Dictionnaires électroniques et analyse automatique de textes. Le système INTEX*. Masson, Paris. 234 p.

[Silberztein, 1994] SILBERZTEIN, M. (1994). INTEX : a corpus processing system. *In Proceedings of the 15th International Conference on Computational Linguistics (COLING'94)*, volume 1, pages 579–583. http ://acl.ldc.upenn.edu/C/C94/C94-1095.pdf.

[Silberztein, 2003] SILBERZTEIN, M. (2003). Finite-state description of the french determiner system. *French Language Studies*, 13:21–246.

[Silberztein, 2004] SILBERZTEIN, M. (2004). Reconnaissance des déterminants français. *LIS*, 24:589–600.

[van Noord et Gerdemann, 2001] van NOORD, G. et GERDEMANN, D. (2001). Finite state transducers with predicates and identities. *Grammars*, 4(3):263–286.

[Vijay-Shanker, 1987] VIJAY-SHANKER, K. (1987). *A Study of Tree Adjoining Grammar*. Thèse de doctorat, University of Pennsylvania, Philadelphia.

[Vivès, 1983] VIVÈS, R. (1983). *Avoir, prendre, perdre : constructions à verbe support et extensions aspectuelles*. Thèse de doctorat, LADL, Université Paris 7.

[Watrin, 2006] WATRIN, P. (2006). *Une approche hybride de l'extraction d'informations : Sous-languages et lexique-grammaire*. Thèse de doctorat, CENTAL, Université Catholique de Louvain-la-Neuve.

[Woods, 1970] WOODS, W. A. (1970). Transition network grammars for natural language analysis. *Communications of the ACM*, 13(10):591–606.

[Younger, 1967] YOUNGER, D. H. (1967). Recognition of context-free languages in time n^3. *Information and Control*, 10(2):189–208.